НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено

Завідувач кафедри

_____ О.В. Коваль
(підпис) (ініціали, прізвище)

“___”_____2019р.

ДИПЛОМНА РОБОТА
на здобуття ступеня бакалавра

з напряму підготовки 6.050103 “Програмна інженерія”

на тему “Система розпізнавання дорожніх знаків на основі нейронної мережі”

Виконав: студент IV курсу, групи ТІ-51

Фастовець Євгеній Русланович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент, Ходаківський Олексій Володимирович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент_____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ — 2019

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший , бакалаврський

Напрям підготовки 6.050103 “ Програмна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2019р.

ЗАВДАННЯ
на дипломну роботу студенту
Фастовцю Євгенію Руслановичу
(прізвище, ім’я, по батькові)

1. Тема роботи “Система розпізнавання дорожніх знаків на основі нейронної мережі”
керівник роботи Ходаківський Олексій Володимирович, доцент
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)
затверджена наказом вищого навчального закладу від 22.05.2019р. № 1325-с
2. Строк подання студентом роботи 10 червня 2019р.
3. Вихідні дані до роботи розроблений модуль написаний мовою С++

4.Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити) виконати аналіз існуючих систем розпізнавання дорожніх знаків за допомогою нейронної мережі, здійснити програмну реалізацію спроектованого модулю, провести тестування і налагодження застосунку.

5. Перелік ілюстративного матеріалу титульний аркуш, поставлені задачі, функції розроблюваної системи, архітектура системи, архітектура нейронної мережі, датасет, imgaug, знак стоп, зображення з датасету, процес навчання нейронної мережі, екран мобільного додатку, екран мобільного додатку після розпізнавання, детальний опис розпізнаного дорожнього знаку, висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ”_14_”___вересня_____2018__р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Вивчення та аналіз задачі	12.02.19	
2	Розробка архітектури та загальної структури системи	23.04.19	
3.	Генерація датасету та тренування нейронної мережі	30.04.19	
4.	Програмна реалізація системи	02.05.19	
5.	Оформлення пояснювальної записки	10.05.19	
6.	Захист програмного продукту	18.05.19	
7.	Передзахист	01.06.19	
8.	Захист	19.06.19	

Студент _____ Фастовець Є. Р.____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ ____доцент, Ходаківський О. В.___
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Дипломну роботу виконано на 69 аркушах, вона містить 2 додатки та перелік посилань на використані джерела з 7 найменувань. У роботі наведено 28 рисунків та 6 таблиць. Виконано реалізацію системи розпізнавання дорожніх знаків з допомогою нейронної мережі. Тема роботи на сьогодніщній день є актуальною, тому що сфера Automotive дуже стрімко розвивається.

Програмний продукт являє собою програму для ПК з використанням нейронної мережі, реалізованої за допомогою фреймворка YOLO, і клієнтський додаток для Android. Обидва програмних продукта створені за допомогою фрейворку Qt, з використанням C++ та QML.

Ключові слова: YOLO, Android, нейронна мережа, C++, Qt, QML.

ABSTRACT

The thesis is completed on 69 sheets, it contains 2 applications and a list of references to used sources of 7 titles. The paper contains 28 figures and 6 tables. Implementation of the system of recognition of road signs with the help of a neural network. The theme of work for the present day is relevant, because the sphere Automotive is very rapidly developing.

The software is a program for a PC using a neural network implemented with the YOLO framework and a client application for Android. Both software products are created using Qt, using C ++ and QML.

Keywords: YOLO, Android, Neural Network, C ++, Qt, QML.

ЗМІСТ

АНОТАЦІЯ 4
ABSTRACT 4
ЗМІСТ 5
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 7
ВСТУП 8
1. СИСТЕМА РОЗПІЗНАВАННЯ ДОРОЖНІХ ЗНАКІВ НА ОСНОВІ НЕЙРОННОЇ МЕРЕЖІ 9
2. ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ РОЗПІЗНАВАННЯ ДОРОЖНІХ ЗНАКІВ 10
2.1. Аналіз існуючих програмних засобів 10
2.2. Огляд мобільного додатку “Traffic Sign Detector” 11
2.3. Classify Traffic Signs[7] 13
2.4. Висновки до розділу 16
3. ЗАСОБИ РЕАЛІЗАЦІЇ ПРОГРАМНОЇ СИСТЕМИ 17
3.1. Вибір архітектури програмного комплексу 17
3.2. Опис архітектури серверу 18
3.3. Опис архітектури клієнтського застосунку 20
3.4. Опис архітектури нейронної мережі 21
3.5. Опис інструментів розробки 22
3.6. Обґрунтування вибору програмної реалізації 28
3.7. Висновки до розділу 29
4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ 30
4.1. Опис функціональності системи 31
4.2. Структура нейронної мережі 32
4.3. Опис створення датасету для нейронної мережі 37
4.4. Процес навчання нейронної мережі 41
4.6. Висновки до розділу 44
5. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ 45

5.1. Інсталяція та системні вимоги 45
5.2. Інструкція з використання програмного продукту 45
ВИСНОВКИ 49
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 50
ДОДАТОК А 51
ДОДАТОК Б 53
ДОДАТОК В 62

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ — програмне забезпечення.

ПК — персональний комп’ютер.

ОС — операційна система.

DL — Deep Learning(глибоке навчання).

ML — Machine Learning(машинне навчання).

ПДД — правила дорожнього руху в Україні.

API (англ. Application Programming Interface) – набір готових класів, процедур, функцій, структур і констант, що надаються додатком (бібліотекою, сервісом) або операційною системою для використання у зовнішніх програмних продуктах

GUI (англ. graphical user interface) – різновид призначеного для користувача інтерфейсу, в якому елементи інтерфейсу (меню, кнопки, значки, списки і т. п.), представлені користувачеві на дисплеї, виконані у вигляді графічних зображень

ВСТУП

За останні роки сфера ML та DL стрімко набирають популярність. Сфера їх застосування стає настільки різноманітною, що іноді важко уявити. Від класифікації зображень до імітації різних живих процесів.

Найближчим часом ці сфери будуть стрімко розвиватись, тому спрогнозувати, що нас чекає в найближчі роки неможливо, але скоріше за все нас чекають прориви в сфері штучного інтелекту.

Зараз існує багато різноманітних алгоритмів, архітектур, які дозволяють створювати ефективні нейронні мережі. Для полегшення створення останніх, вже створено тисячі різноманітних бібліотек і фреймворків.

Однією з сфер, в якій нейронні мережі набирають популярність є автомобільна сфера (Automotive). Автопілот є однією з передових технологій, якому необхідні різноманітні нейронні мережі, для аналізу навколишнього середовища з різних джерел інформації (датчики, камери, і т.п.).

Одним із таких джерел є відео з камер. Більша частина із цієї інформації, на перший погляд, є несуттєвою, але з неї можна витягнути більше користі ніж здається.

Тому, було запропоновано дослідити можливість реалізувати систему для розпізнавання дорожніх знаків та інформувати користувачу, дію цього знаку.

1. СИСТЕМА РОЗПІЗНАВАННЯ ДОРОЖНІХ ЗНАКІВ НА ОСНОВІ НЕЙРОННОЇ МЕРЕЖІ

У сучасному світі все більше людської діяльності автоматизується. Але існує багато сфер людської діяльності де їхня праця застосовується тільки тому, що її важко автоматизувати. Стрімкий розвиток DL надає нові величезні можливості для застосування комп'ютерів замість людей в певній діяльності, яку комп'ютер виконає з меншими помилками і більшою ефективністю роботи.

Серед таких діяльностей можна виділити керування автомобілем. Тут велике значення має людський фактор, людина може запросто відволіктися або просто не звернути увагу на дорожній знак, розмітку дороги, іншого учасника руху. Також людина не може одночасно оброблювати велику кількість інформації. Наприклад, перехрестя на якому розташовано дуже багато різноманітних дорожніх знаків.

Тому було запропоновано дослідити можливість реалізувати систему для розпізнавання дорожніх знаків та інформувати користувачу дію цього знаку.

Як результат дослідження, потрібно створити програмний застосунок, основними функціями якого є:

- Забезпечення користувача додатком на смартфон, в якому він може фотографувати і розпізнавати дорожні знаки.
- Розпізнавання дорожніх знаків.
- Класифікація дорожніх знаків.
- Сповіщення користувача про дорожні знаки, які було розпізнано на зображенні.

Забезпечити користувача зручним графічним інтерфейсом. При розробці необхідно звернути увагу на надійність системи, простоту експлуатації та її вартість.

Робота буде виконуватись на операційній системі Linux. Для розробки системи обраний фреймворк Qt, який є кросплатформеним та досить поширеним в сфері Automotive. Додаток буде запрограмований на мові C++ (серверна частина) та QML (інтерфейс користувача).

2. ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ РОЗПІЗНАВАННЯ ДОРОЖНІХ ЗНАКІВ

З кожним роком все більше людської діяльності автоматизується. Так керуванням сучасними автомобілями може займатись не людина, а автопілот. Він використовує різноманітне програмне забезпечення, яке основане на різноманітних алгоритмах машинного та глибокого навчання. Серед них можна виділити і задачу розпізнавання дорожніх знаків на різних зображень.

Тому було вирішено розробити програмну систему, яка допоможе розпізнавати дорожні знаки на зображеннях. В майбутньому таку систему можна буде інтегрувати до різних автопілотів.

2.1. Аналіз існуючих програмних засобів

У процесі пошуку інформації та аналізу існуючих рішень, було виявлено, що на даний момент існують системи для розпізнавання знаків закордонних країн та не в повному обсязі:

— мобільний додаток на Andorid “Traffic Sign Detector” — даний додаток має декілька недоліків, а саме це достатньо велика похибка під час розпізнавання та непридатність для розпізнавання дорожніх знаків, які розташовані в Україні.

Також є системи які не мають програмного забезпечення, а складаються лише з нейронної мережі. Ці системи розраховані для розробників і для звичайних користувачів не несуть ніякої цінності. Вони побудовані на різних фреймворках, але ні один з них не написаний мовою C/C++. А ця мова є пріоритетною в автомобільній розробці.

Тому створення системи розпізнавання дорожніх знаків за допомогою нейронної мережі на мові C++ є актуальною задачею.

2.2. Огляд мобільного додатку “Traffic Sign Detector”

Даний мобільний додаток виявляє та оголошує дорожні знаки під час руху. Для розпізнавання використовується камера та нейронна мережа.

Основні можливості:

- виявляє дорожні знаки та світлофори за допомогою камери;
- оголошує, що було виявлено;
- зберігає історію виявлення для подальшого перегляду.

На рисунку 2.1 можна побачити основне вікно програми:

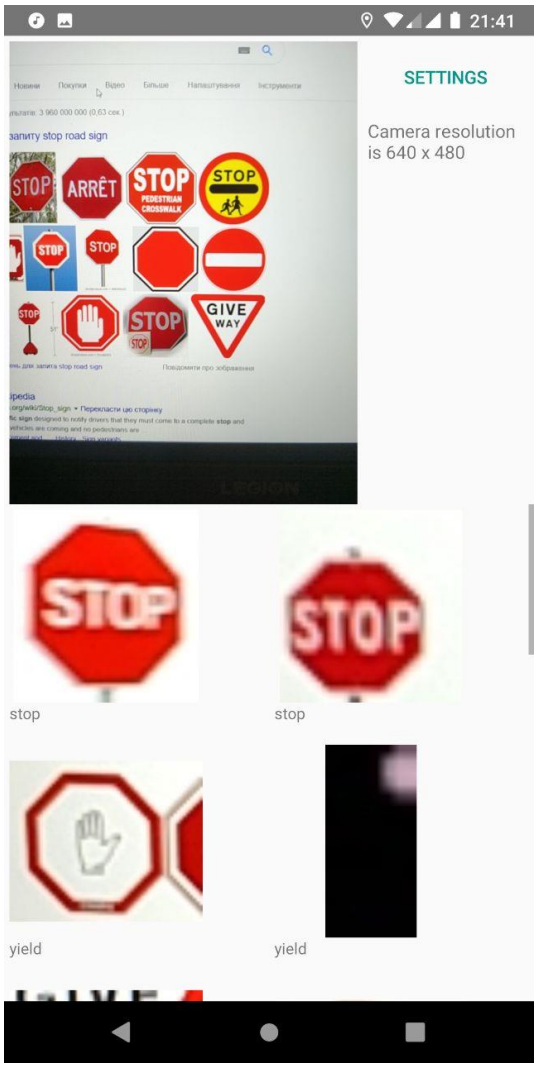

Рисунок 2.1 – Інтерфейс додатку Traffic Sign Detector

Нижче (рисунок 2.2) наведено приклад поведінки програми, коли до кадру потрапляє звичайний коробок сірників. Програма починає знаходити знаки там де їх і немає. Така поведінка не припустима в таких відповідальних додатках.

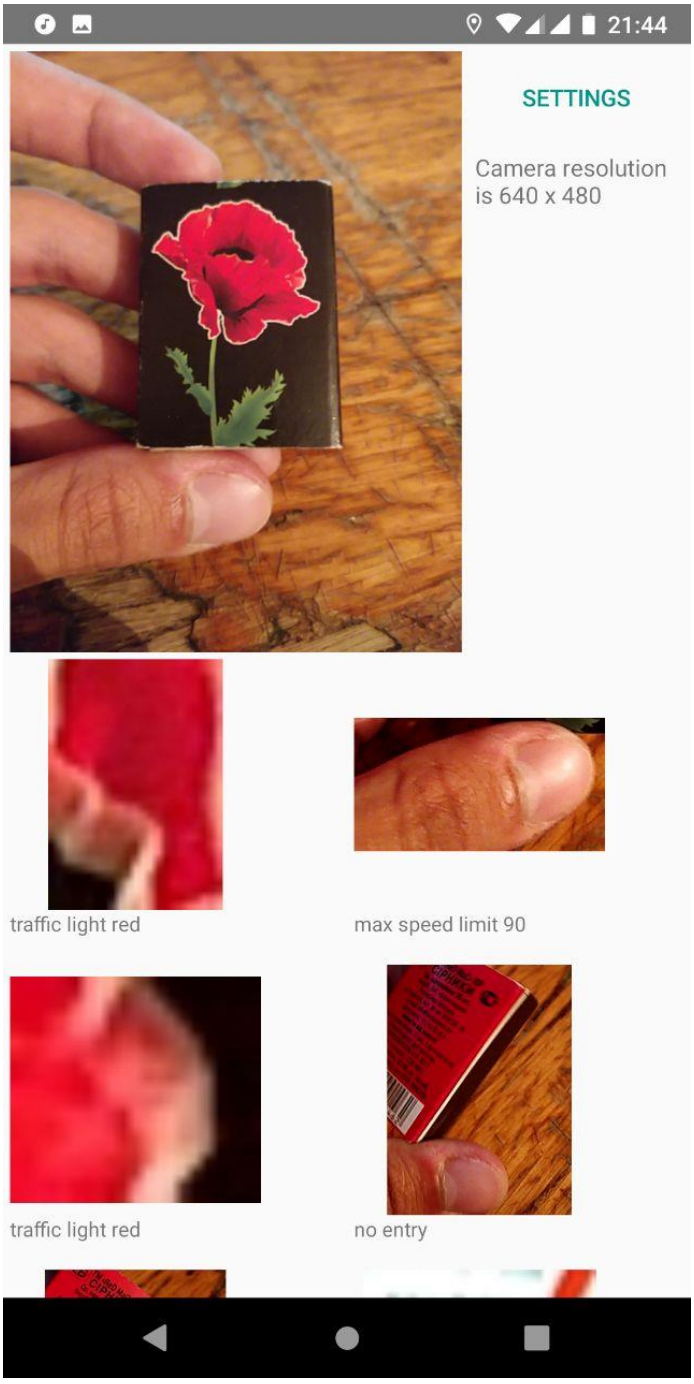

Рисунок 2.2 — Хибні розпізнавання

Серед “розпізнаних” знаків можна бачити обмеження по швидкості та в’їзд заборонено.

2.3. Classify Traffic Signs[7]

Запропонована архітектура натхненна роботою Yann Le Cun щодо класифікації дорожніх знаків. Було додано кілька налаштувань і створено модульну кодову базу, яка дозволяє випробувати різні розміри фільтрів, глибину і кількість шарів згортки, а також розміри повноз’єднаних шарів.

В основному в даній роботі спробували розміри фільтрів 5x5 і 3x3 (ядра ядра) і почали з глибини 32 для першого згорткового шару. На рисунку 2.3 показано архітектуру 3x3 EdLeNet

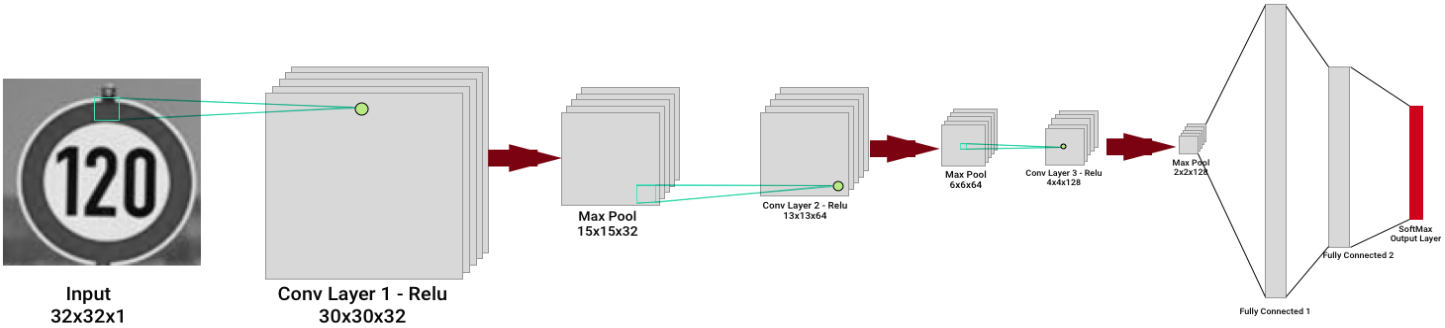

Рисунок 2.3 — Архітектура EdLeNet 3x3

Мережа складається з 3 згорткових шарів — розмір ядра 3x3, з подвоєнням глибини на наступному шарі — з використанням ReLU як функції активації, кожна з яких супроводжується операцією об'єднання 2x2 max. Останні три шари повноз'єднані, а остаточний шар — 43 результати (загальна кількість можливих міток), обчислена за допомогою функції активації SoftMax. Мережа навчається з використанням міні-серійного стохастичного градієнтного спуску з оптимізатором Адама. Ми створюємо високомодульну інфраструктуру кодування, що дозволяє нам динамічно створювати наші моделі, як у таких фрагментах:

```
mc_3x3 = ModelConfig(EdLeNet,
"EdLeNet_Norm_Grayscale_3x3_Dropout_0.50", [32, 32, 1], [3, 32, 3],
[120, 84], n_classes, [0.75, 0.5])
mc_5x5 = ModelConfig(EdLeNet,
```

```
"EdLeNet_Norm_Grayscale_5x5_Dropout_0.50", [32, 32, 1], [5, 32, 2],
[120, 84], n_classes, [0.75, 0.5])

me_g_norm_drpt_0_50_3x3 = ModelExecutor(mc_3x3)
me_g_norm_drpt_0_50_5x5 = ModelExecutor(mc_5x5)
```

ModelConfig містить інформацію про модель, таку як:

- функція моделі (наприклад, EdLeNet);
- назву моделі;
- формат введення (наприклад, [32, 32, 1] для відтінків сірого);
- конфігурація згорткових шарів [розмір фільтра, глибина початку, кількість шарів];
- розміри повністю з'єднаних шарів (наприклад, [120, 84]);
- кількість класів;
- відсівання зберігають відсоткові значення [p-conv, p-fc].

ModelExecutor відповідає за підготовку, оцінку, прогнозування та створення візуалізацій карти активації.

Щоб краще ізолювати моделі та переконатися, що вони не всі існують під одним графіком Tensorflow, тут було використовано наступну корисну конструкцію:

```
self.graph = tf.Graph()
with self.graph.as_default() as g:
   with g.name_scope( self.model_config.name ) as scope:
...
with tf.Session(graph = self.graph) as sess:
```

Таким чином, створювались окремі графіки для кожної моделі, переконавшись, що немає змішування змінних, заповнювачів тощо.

Найкращі результати нейроної мережі були отримані з глибиною 32. Також тут порівнюються кольорові та сірі тони, стандартні та нормалізовані зображення. Також була помічена деяка нестійка поведінка втрат на валідаційному наборі після заданого числа епох, що фактично означало, що наша модель перенасичувалась на

тренувальному наборі і не навчалась. Нижче наведено деякі метричні графіки для різних модельних конфігурацій.

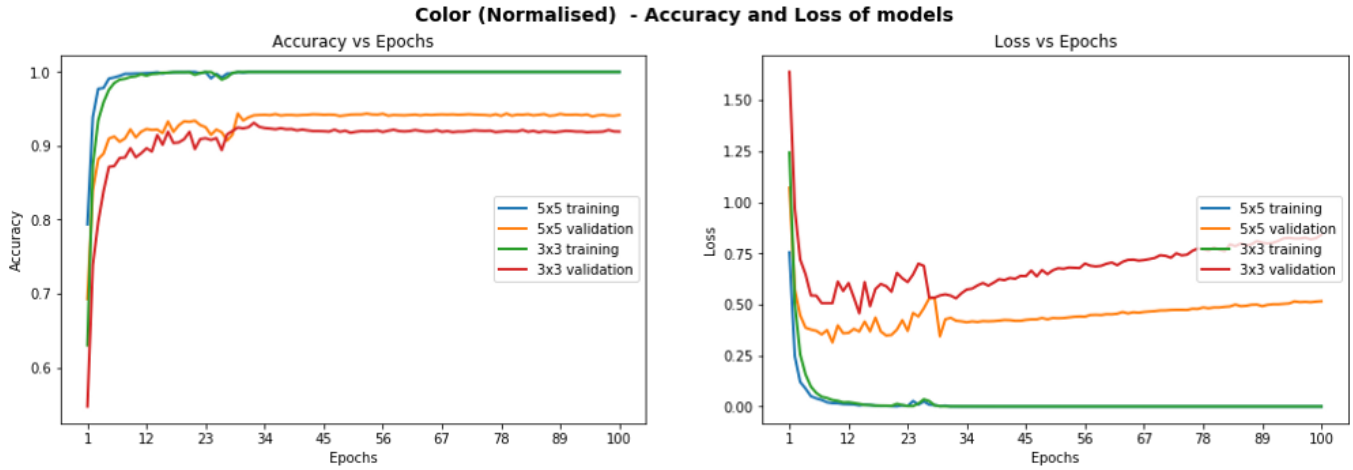

Рисунок 2.4 — Продуктивність моделей на кольорових нормалізованих зображеннях

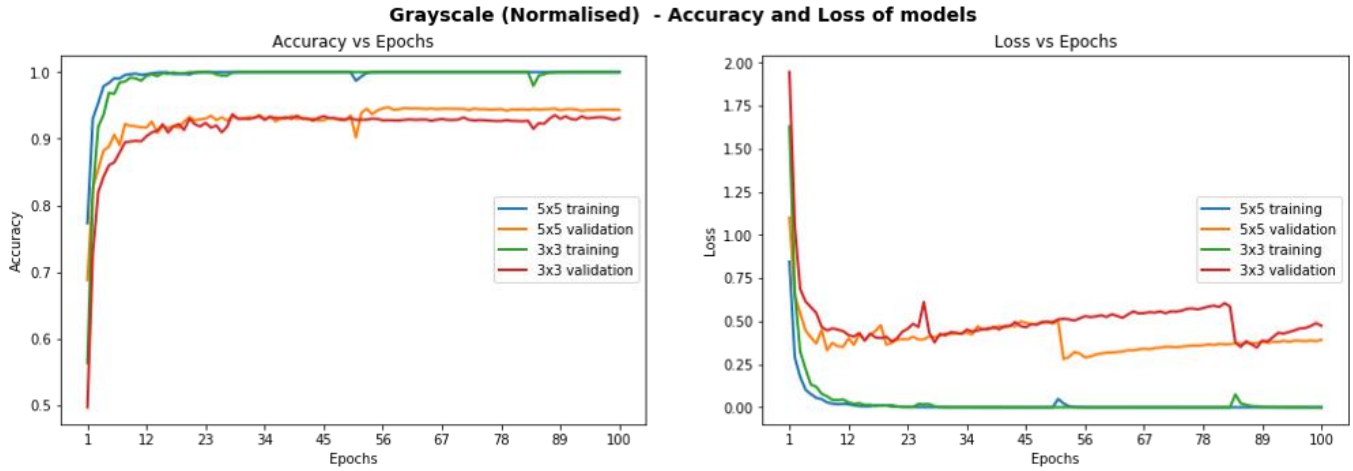

Рисунок 2.5 — Продуктивність моделей на нормалізованих зображеннях у градаціях сірого

Дана система була розроблена за допомогою глибинного навчання для класифікації дорожніх знаків з високою точністю, з використанням різноманітних методів попередньої обробки і регуляризації (наприклад, відсіву), а також спроби різних архітектур моделі. Модель досягла близького до 98%. Але не зважаючи на таку точність, ця система не є пригодною для використання на території України, тому що навчання відбувалося на наборі знаків, які не відповідають знакам, які затвердженні

в ПДД.

У майбутньому, я вважаю, що дана система може збільшити свою точність та розширити датасет для розпізнвання в інших країнах.

2.4. Висновки до розділу

Було роглянуто основні системи розпізнавання дорожніх знаків. Майже всі системи розроблені не на мові C++ та не для українського ринку. Також було розглянуто додаток на Android, який має досить велику похибку під час розпізнавання.

3. ЗАСОБИ РЕАЛІЗАЦІЇ ПРОГРАМНОЇ СИСТЕМИ

Аналізуючи поставлену задачу та методи її вирішення, було вирішено розроблювати програмний комплекс на основі двох фреймворків Qt та Yolo[2]. Перший кросплатформений, широко відомий в сфері Automotive. Другий реалізований на мові C, що дозволяє використовувати всі ресурси пристрою на максимум.

3.1. Вибір архітектури програмного комплексу

Для реалізації поставленої задачі було вирішено використовувати архітектуру, яка складається з таких компонентів: сервер та клієнт. Спілкуються вони за допомогою технології websocket. Схема даної архітектури зображена на рисунку 3.1.

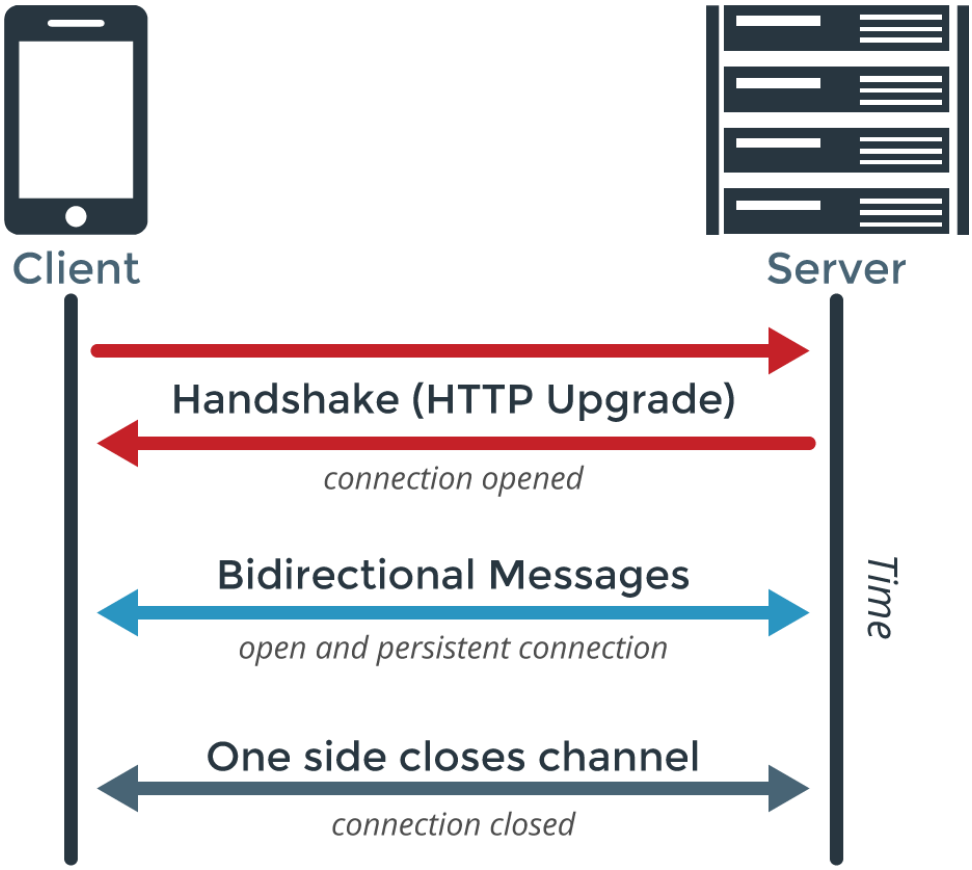

Рисунок 3.1 – Архітектура програмного комплексу

Головним центром програмного комплексу є сервер. У ньому зосереджена основна бізнес-логіка. За допомогою серверу відбувається розпізнавання дорожніх знаків за допомогою нейронної мережі. На сервері знаходяться конфігураційний файл, список знаків які можна розпізнати та файл з вагами для нейронної мережі Також тут відбуваються всі позначення розпізнаних знаків на зображенні.

Під час користування програмою, користувач взаємодіє з клієнтом, який створений у вигляді додатку на Android. В ньому реалізований інтерфейс, за допомогою якого користувач може зробити фото та отримати результат розпізнавання, яке відбувається на стороні серверу.

3.2. Опис архітектури серверу

Шаблон проектування — це архітектура, рішення яке описує яким чином вирішуються задачі, які часто зустрічаються при розробці програмних систем чи додатків.

Для реалізації серверу було використано фреймворк Qt. Та реалізовано шаблон проектування MVC.

MVC (Model-View-Controller) — це шаблон проектування, головною ідеєю якого є відділити логіку застосунку від представлення (рисунок 3.2). Принципом MVC є розділення програмної реалізації системи на три головні компоненти: M — Model (Модель), V — View (Представлення), C — Controller (Контролер), таким чином, що редагування будь-якого компонента може відбуватися незалежно.

Центральним компонентом шаблону MVC є модель. Вона є незалежною від користувацького інтерфейсу. Модель повинна лише керувати даними, бізнес логікою та основними правилами поведінки застосунку.

Представлення — це відображення будь-якої інформації, одержаної на виході, наприклад графік чи діаграма. Одні й ті самі дані можуть бути зображені на декількох представленнях одночасно, наприклад гістограма для керівництва компанії й таблиці для бухгалтерії. Представлення — це кінцевий інтерфейс, з яким взаємодіє

користувач. Користувач може передавати дані через представлення.

Контролер — реалізує взаємодію між моделлю і представленням. Він обробляє вхідні дані й перетворює їх на команди для моделі чи вигляду.

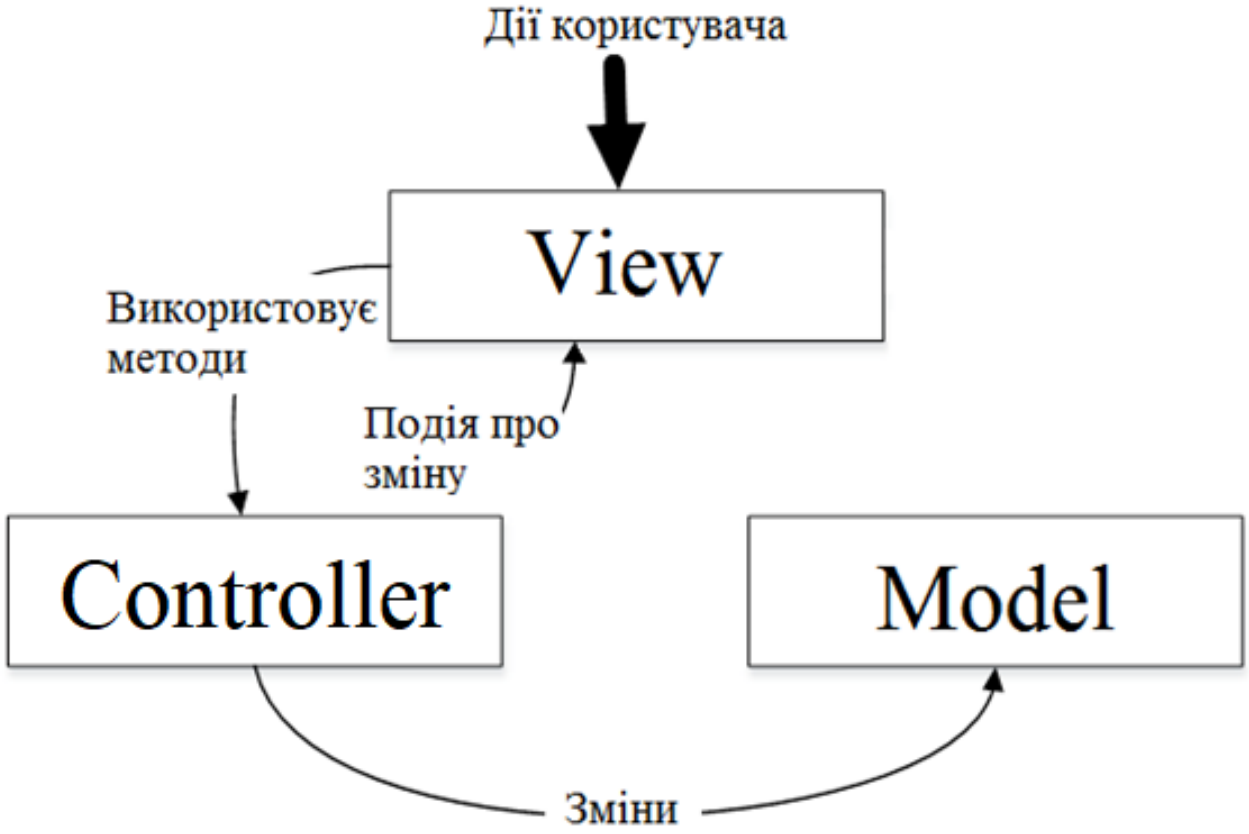

Рисунок 3.2 – Схема роботи MVC шаблону

Мета шаблону — гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах дозволяє застосувати тестування різноманітних компонентів програмної системи та сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності.

3.3. Опис архітектури клієнтського застосунку

Для реалізації клієнтського додатку також було використано фреймворк Qt, а саме технологію QML.

QML — є мовою розмітки інтерфейсу користувача. Це декларативна мова (подібно до CSS і JSON) для розробки додатків, орієнтованих на інтерфейс користувача. Модулі QML, що постачаються з Qt, включають примітивні графічні блоки (наприклад, Rectangle, Image)(рисунок 3.3).

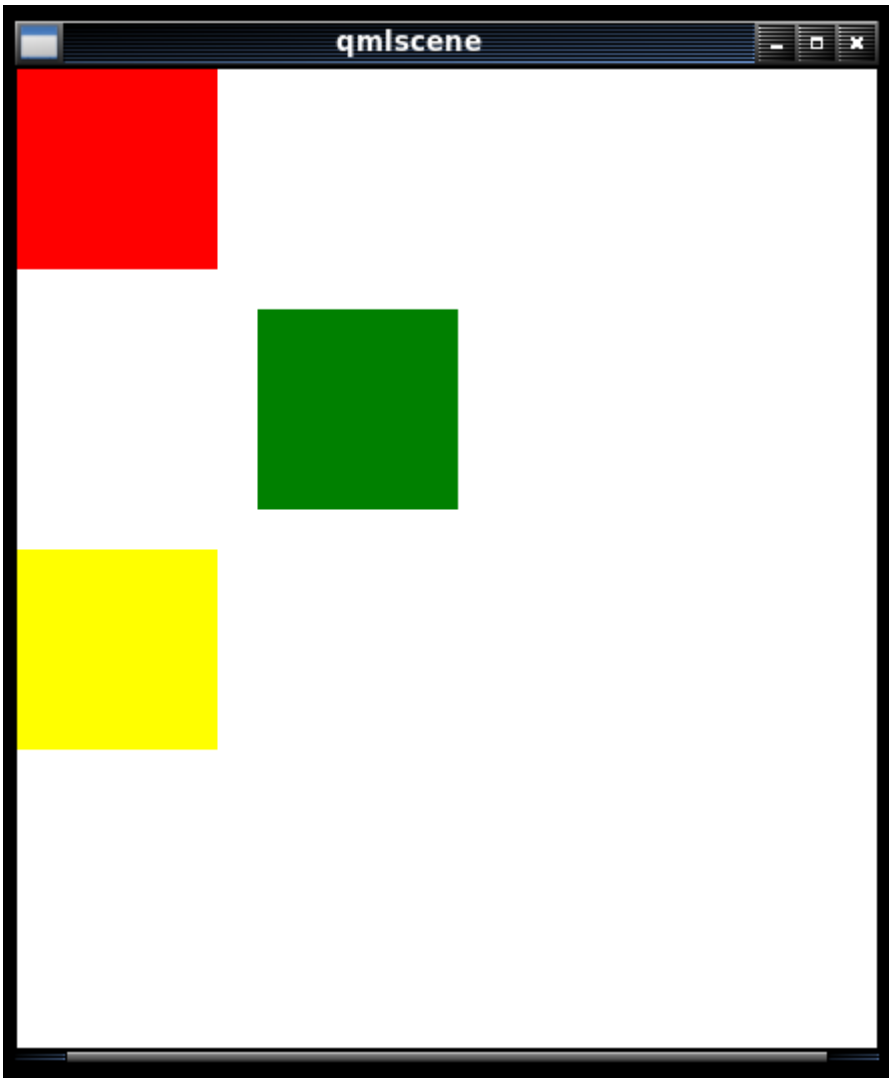

Рисунок 3.3 — Приклад компоненту Rectangle

Компоненти моделювання (наприклад, FolderListModel, XmlListModel), поведінкові компоненти (наприклад, TapHandler, DragHandler, State, Transition, Animation) і більш складні елементи керування (наприклад, кнопка, повзунок, меню).

А також різні компоненти для передачі даних через мережу. Одним з таких є WebSocket. З його допомогою відбувається відправлення даних до серверу.

Елементи QML можуть бути доповнені стандартним JavaScript так і через включені файли .js. Елементи також можуть бути легко інтегровані і розширені за допомогою компонентів C ++, використовуючи фреймворк Qt.

Клієнтський додаток дуже схожий з додатком камери. По натисканню кнопки створюється знімок, який відправляється на сервер, потім приходить у відповідь знімок уже з розпізнаним дорожнім знаком.

3.4. Опис архітектури нейронної мережі

Для реалізації нейронної мережі було використано фреймворк Yolo[2].

Системи попереднього виявлення, на відміну від Yolo[2], повторюють класифікатори або локалізатори для виявлення певних об’єктів. Вони застосовують модель до зображення в різних місцях і масштабах. Високі скориновані області зображення вважаються розпізнаванням певного об’єкту на зображенні.

Фреймворк Yolo[2] використовує зовсім інший підхід. Він застосовує єдину нейронну мережу до повного зображення. Ця мережа ділить зображення на регіони і передбачає обмежувальні рамки та ймовірності для кожного регіону. Ці обмежувальні рамки зважуються за прогнозованими ймовірностями.

Ця модель має ряд переваг перед системами на основі класифікаторів. Вона дивиться на все зображення під час тестування, тому її прогнози повідомляються глобальним контекстом у зображенні. Вона також робить прогнози з єдиною мережевою оцінкою на відміну від систем, таких як R-CNN, які вимагають тисячі для одного зображення. Це робить її надзвичайно швидкою (рисунок 3.4).

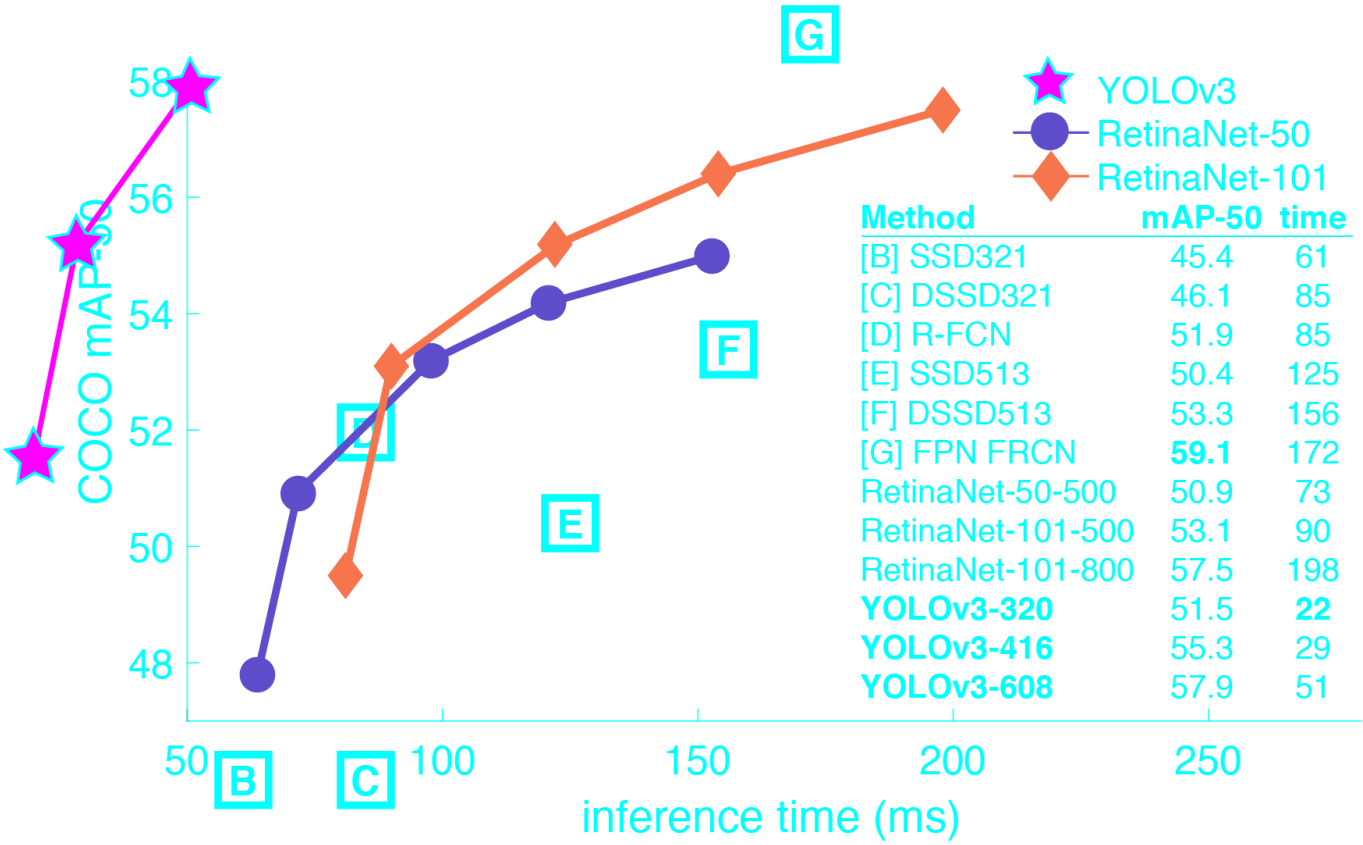

Рисунок 3.4 — Порівняння YOLO з конкурентами

Як видно з рисунку YOLO більш ніж в 1000 разів швидше, ніж R-CNN і в 100 разів швидше, ніж швидкий R-CNN.

3.5. Опис інструментів розробки

Програмний комплекс побудований з використанням різних технологій і головною ціллю було створення доступного і зручного рішення з використанням відкритих технологій.

Для створення клієнтського та серверного додатку було використано такий набір технологій:

Qt Creator — це комплексне середовище розробки C ++, JavaScript і QML, що є частиною SDK для середовища розробки додатків Qt GUI. Вона включає в себе візуальний відладчик і вбудований графічний інтерфейс і дизайнер форм. Функції

редактора включають підсвічування синтаксису та автозавершення. Qt Creator використовує компілятор C++ з колекції компіляторів GNU на Linux і FreeBSD. У Windows вона може використовувати MinGW або MSVC із встановленою за замовчуванням установкою, а також може використовувати Microsoft Console Debugger при компіляції з вихідного коду. Clang також підтримується. Інтерфейс програми зображений на рисунку 3.5.

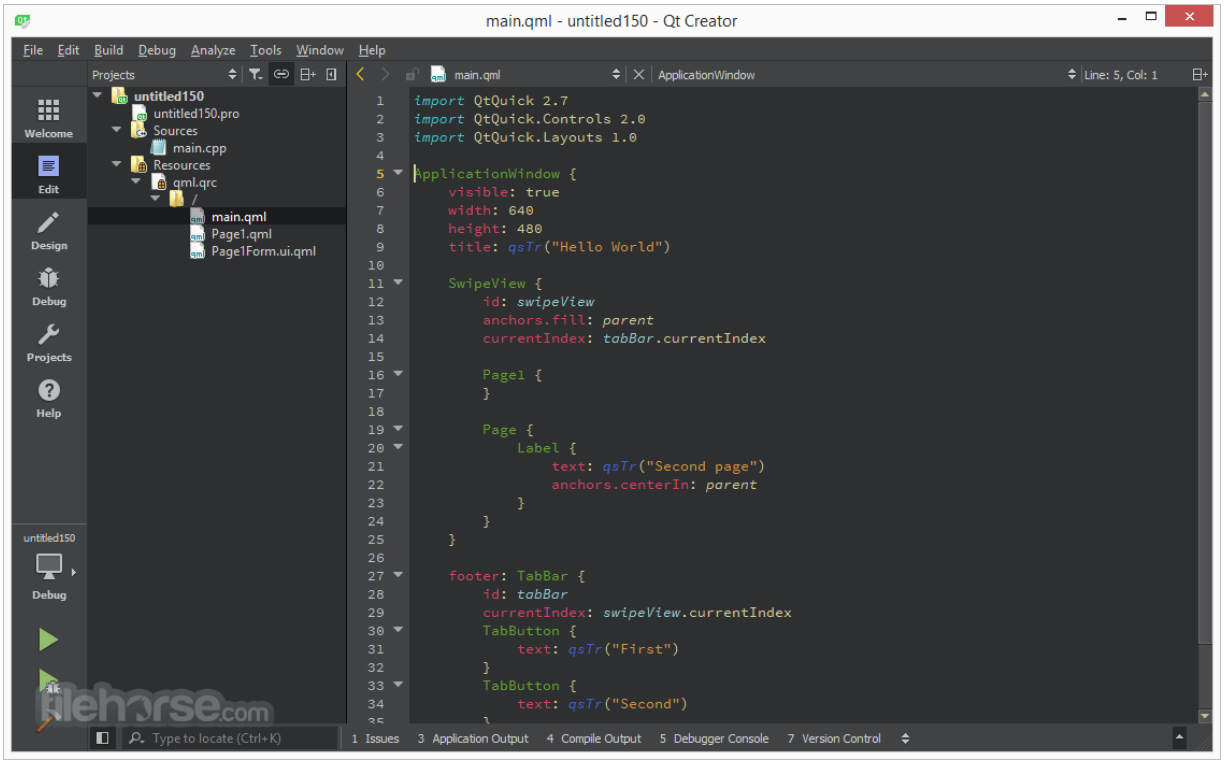

Рисунок 3.5 — Середовище розробки QtCreator

Qt — це безкоштовний і відкритий інструментарій для створення графічних користувацьких інтерфейсів, а також крос-платформних програм, які працюють на різних програмних і апаратних платформах, таких як Linux, Windows , macOS, Android або вбудованих системах. В даний час Qt розробляється компанією Qt, публічно зареєстрованою компанією, і окремими розробниками та організаціями, що працюють над просуванням Qt. Він доступний як для комерційних ліцензій, так і для ліцензій GPL 2.0, GPL 3.0 і LGPL 3.0.

Qt підтримує різні компілятори, включаючи компілятор GCC C ++ і набір Visual Studio і має широку підтримку інтернаціоналізації. Qt також надає Qt Quick, що

включає декларативну мову сценаріїв QML, що дозволяє використовувати JavaScript для забезпечення логіки. За допомогою Qt Quick стала можлива швидка розробка додатків для мобільних пристроїв, в якому логіка все ще може бути написана на мові C++.

Інші функції включають доступ до бази даних SQL, синтаксичний аналіз XML, синтаксичний аналіз JSON, управління потоками та підтримку мережі.

QML (Qt Modeling Language) — є мовою розмітки інтерфейсу користувача. Це декларативна мова (подібно до CSS і JSON) для розробки додатків, орієнтованих на інтерфейс користувача. Вона пов'язана з Qt Quick — комплектом створення інтерфейсу користувача, спочатку розробленим компанією Nokia у рамках Qt.

Qt Quick часто використовується для мобільних додатків. QML також використовується з Qt3D для опису 3D-сцени. Документ QML описує ієрархічне дерево об'єктів. Модулі QML, що постачаються з Qt, включають примітивні графічні блоки (наприклад, Rectangle, Image), компоненти моделювання (наприклад, FolderListModel, XmlListModel), поведінкові компоненти (наприклад, TapHandler, DragHandler, State, Transition, Animation) і більш складні елементи керування (наприклад, кнопка, повзунок, меню). Ці елементи можуть бути об'єднані для побудови компонентів різної складності, від простих кнопок і повзунків, до завершення програм з підтримкою Інтернету.

Елементи QML можуть бути доповнені стандартним JavaScript так і через включені файли .js. Елементи також можуть бути легко інтегровані і розширені за допомогою компонентів C ++, використовуючи фреймворк Qt.

Коди QML і JavaScript можуть бути скомпільовані у бінарні файли C ++ за допомогою Qt Quick Compiler. Крім того, існує формат файлу кешу QML, який динамічно зберігає скомпільовану версію QML для швидшого запуску наступного разу.

Websocket — це протокол, що призначений для обміну інформацією між клієнтом та веб-сервером в режимі реального часу. Він забезпечує двонаправлений повнодуплексний канал зв'язку через один TCP-сокет. Websocket спроектовано для втілення у веб-браузерах та веб-серверах, але може також використовуватись будь-

яким клієнт-серверним застосунком. Прикладний програмний інтерфейс websocket був стандартизований W3C, крім того протокол websocket стандартизований IETF як RFC 6455.

Для створення та тренування нейронної мережі було використано такий набір технологій:

Python — є інтерпретованою мовою програмування високого рівня загального призначення. Створений Гвідо ван Россумом і вперше випущений в 1991 році, філософія дизайну Python підкреслює читабельність коду з його помітним використанням значних пробілів. Його мовні конструкції та об'єктно-орієнтований підхід спрямовані на те, щоб допомогти програмістам написати чіткий, логічний код для малих і великих проектів.

Python динамічно типізований. Він підтримує багато парадигм програмування, включаючи процедурні, об'єктно-орієнтовані та функціональні програми.

Функції оголошуються наступним чином:

```
def fib(n):
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()
fib(1000),
```

це приклад написання функції, яка рахує числа Фібоначчі.

Python був задуманий в кінці 1980-х років як наступник мови ABC. Python 2.0, випущений 2000 року, були представлені такі функції, як списки розуміння і система збору сміття, здатна збирати еталонні цикли. Python 3.0, випущений у 2008 році, був серйозною ревізією мови, яка не є повністю зворотно сумісною, і багато Python 2-код не працює без змін на Python 3. Через занепокоєння щодо кількості коду, написаного для Python 2, підтримка Python 2.7 (останній реліз в серії 2.x) був розширений до 2020

року. Розробник мови Гвідо ван Россум взяв на себе відповідальність за проект до липня 2018 року, але тепер поділяє його керівництво як член ради з п'яти осіб.

Інтерпретатори Python доступні для багатьох операційних систем. Світове співтовариство програмістів розробляє та підтримує CPython.

OpenCV (Open source computer vision) — це бібліотека функцій програмування, спрямована головним чином на комп'ютерне бачення в реальному часі. Спочатку була розроблена компанією Intel (Рисунок 3.6), пізніше її підтримували Willow Garage, а потім Itseez (який згодом був придбаний Intel). Бібліотека є крос-платформенною і вільною для використання під ліцензією BSD з відкритим вихідним кодом. OpenCV підтримує глибокі рамки навчання TensorFlow, Torch / PyTorch і Caffe.

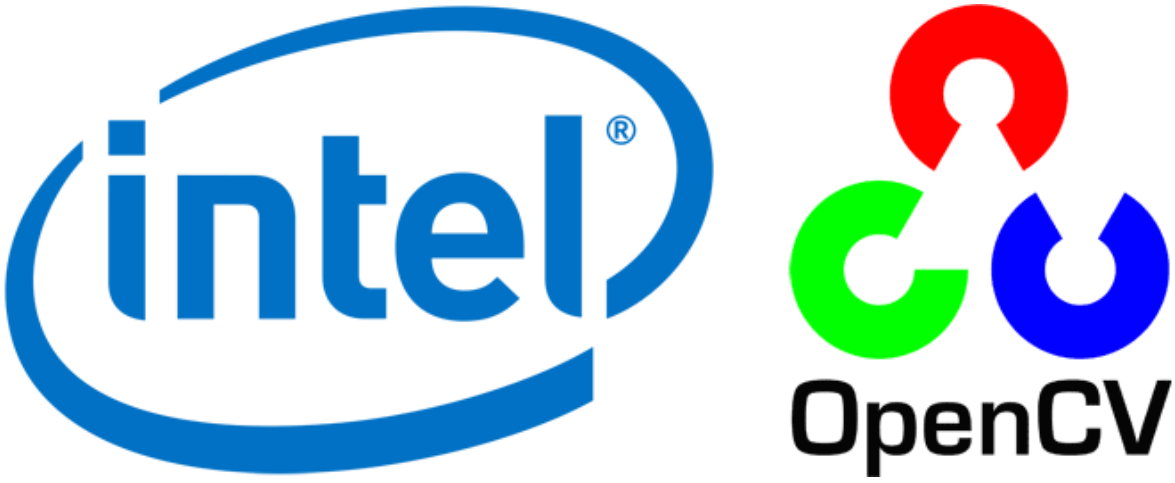

Рисунок 3.6 — OpenCV та Intel

OpenCV написана на мові C ++. Є прив'язки в Python, Java і MATLAB / OCTAVE. API для цих інтерфейсів можна знайти в онлайн-документації. Обгортки на інших мовах, таких як C #, Perl, Ch, Haskell, Ruby, були розроблені для заохочення широкої аудиторії.

Починаючи з версії 3.4, OpenCV.js — це прив'язка JavaScript для обраної підсистеми функцій OpenCV для веб-платформи.

Всі нові розробки та алгоритми в OpenCV тепер розроблені на мові C ++.

CUDA — це паралельна обчислювальна платформа і модель інтерфейсу прикладного програмування (API), створена Nvidia. Це дозволяє розробникам

програмного забезпечення та програмним інженерам використовувати графічний процесор з підтримкою CUDA (GPU) для обробки загального призначення - підхід, що називається GPGPU. Платформа CUDA — це програмний шар, який надає прямий доступ до віртуального набору команд GPU і паралельних обчислювальних елементів для виконання команд на обчислювальних ядрах.

Платформа CUDA призначена для роботи з мовами програмування, такими як C, C++ і Fortran. Ця доступність полегшує для фахівців з паралельного програмування використання ресурсів графічного процесора, на відміну від попередніх API, таких як Direct3D і OpenGL, які вимагали передових навичок у графічному програмуванні. Крім того, CUDA підтримує рамки програмування, такі як OpenACC і OpenCL. Коли вона була вперше представлена Nvidia, назва CUDA була абревіатурою для Compute Unified Device Architecture, але Nvidia згодом відмовилася від використання акроніму.

CUDA має ряд переваг перед традиційними загальноприйнятими обчисленнями на графічних процесорах (GPGPU), використовуючи графічні API:

- Розсіяне читання — код може читатися з довільних адрес в пам'яті;
- Єдина віртуальна пам'ять (CUDA 4.0 і вище);
- Єдина пам'ять (CUDA 6.0 і вище);
- Спільна пам'ять — CUDA надає швидку область спільної пам'яті, яку можна розділити між потоками. Це може використовуватися як кеш користувачем, що дозволяє підвищувати пропускну здатність, ніж це можливо за допомогою пошуку текстур;
- Швидше завантаження та повернення до GPU;
- Повна підтримка цілочисельних і побітових операцій, включаючи цілочисельні пошуки текстури.

VLC — є безкоштовним і відкритим, портативним, крос-платформним медіа-програвачем і потоковим медіа-сервером, розробленим проектом VideoLAN. VLC доступний для настільних операційних систем і мобільних платформ, таких як Android, iOS, Tizen, Windows 10 Mobile і Windows Phone. VLC також доступний на платформах цифрового розповсюдження, таких як Apple App Store, Google Play і

Microsoft Store.

VLC підтримує багато методів стиснення аудіо та відео та форматів файлів, включаючи DVD-Video, відео CD і потокові протоколи. Він може передавати мультимедійні файли через комп'ютерні мережі та перекодувувати мультимедійні файли.

Розподіл за замовчуванням VLC включає в себе велику кількість вільних бібліотек декодування і кодування, уникаючи необхідності пошуку / калібрування власних плагінів. Бібліотека libavcodec з проекту FFmpeg надає багато кодеків VLC, але в основному VLC використовує власні мультиплексори і демультиплексори. Вона також має свої власні реалізації протоколу. Він також отримав відзнаку як перший програвач для підтримки відтворення зашифрованих DVD-дисків на Linux та macOS за допомогою бібліотеки libdvdcss DVD decryption.

3.6. Обґрунтування вибору програмної реалізації

При проектуванні системи було вивчено та проаналізовано предметну область. Проаналізувавши її було вирішено розроблювати програмний продукт, який буде оснований на мові C++ для подальшої можливості вбудувати його в різноманітні автомобільні рішення. А фреймворк Qt тільки покращує можливості кросплатформеності. Таким чином програмний продукт не буде обмежений якоюсь певною платформою.

Використання фреймворку Yolo[2] для створення нейронної мережі є кращим варіантом для даної системи, тому що даний фреймворк написаний мовою C. Це дає нам можливість з легкістю інтегрувати нейронну мережу до будь-якої програми написаної на мові C/C++.

Використання технологій CUDA та opencv значно покращує та пришвидшує процес тренування нейронної мережі та процес розпізнавання дорожніх знаків. OpenCV дозволяє виконувати попередню підготовку зображення перед самим процесом розпізнавання.

Використання мови програмування Python дозволяє за допомогою скриптів генерувати датасет для тренування і валідації нейронної мережі.

Плеєр VLC дозволяє розбивати відео на окремі зображення, які в подальшому використовуються як фон, на який додаються дорожні знаки.

3.7. Висновки до розділу

У даному розділі було виконано аналіз засобів реалізації програмного продукту та розглянуто архітектури серверу та клієнту розроблюваної системи. Було описано весь інструментарій, який використовується для розробки та обгрунтовано вибір тих чи інших технологій.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Система розпізнавання дорожніх знаків складається з додатку на смартфон та ПЗ на комп’ютер. На останньому і буде відбуватися основний процес — розпізнавання дорожніх знаків. За допомогою додатку на смартфон користувач легко та інтуїтивно зможе користуватися системою.

На рисунку 4.1 наведена схема структури системи, на якій розташовані всі програмні модулі.

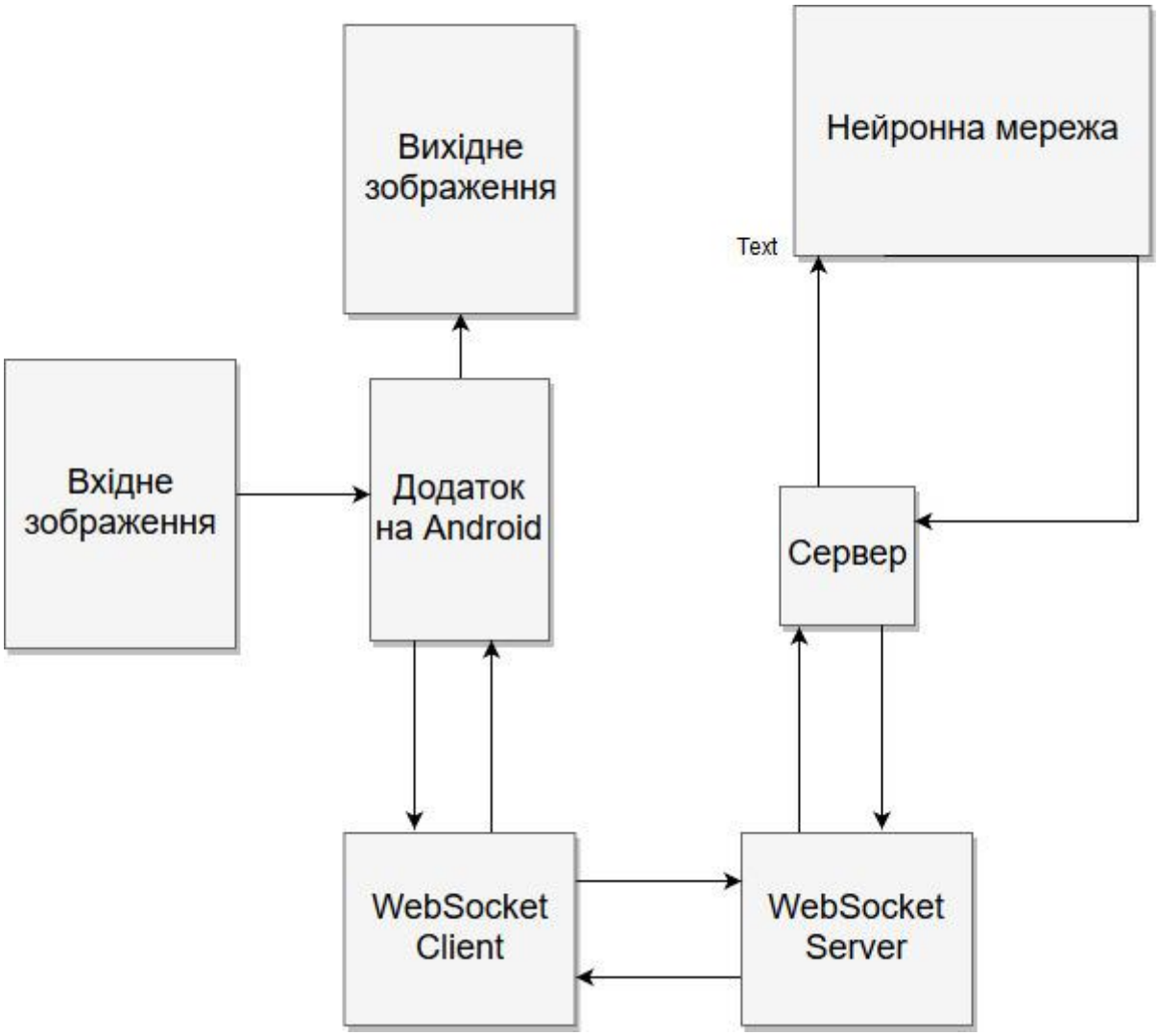

Рисунок 4.1 — Схема структури системи

На сервері відбувається розпізнавання дорожніх знаків за допомогою нейронної мережі. Також тут знаходяться конфігураційний файл, список знаків які можна розпізнати та файл з вагами для нейронної мережі.

На клієнті користувач робить фото і за допомогою WebSocket технології відправляє його на сервер. Потім отримує результат розпізнавання і відображає на екрані.

4.1. Опис функціональності системи

Оскільки даний програмний продукт розрахований на клієнта та не потребує зовнішнього втручання інших осіб, тому дана система містить у собі одного головного актора — користувача системи.

На рисунку 4.2 представлена діаграма прецедентів, яка описує функції та дії актора у системі.

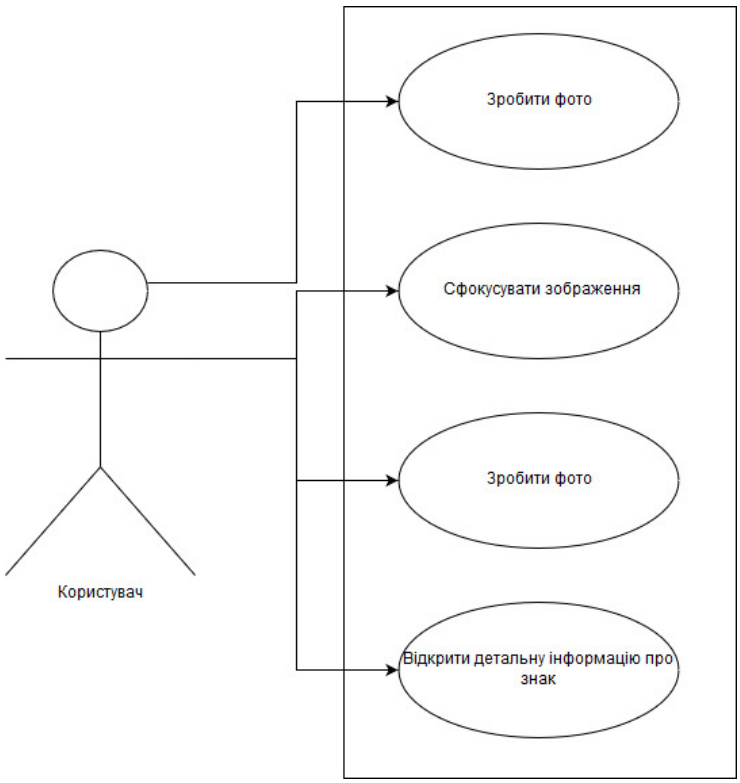

Рисунок 4.2 — Діаграма прецедентів системи

4.2. Структура нейронної мережі

Нейронна мережа складається з детектора Yolo[2], який представлений у вигляді динамічної бібліотеки та трьох файлів:

- road_signs.names — файл в якому знаходиться список дорожніх знаків (класів), які можуть бути розпізнані.
- road-signs-obj-tiny.weights — файл з вагами(різними коефіцієнтами). Завдяки цьому файлу нейронна мережа вибирає той чи інший шлях для розпізнавання.
- road-signs-obj-tiny.cfg — конфігураційний файл, в якому знаходяться всі параметри нейронної мережі та її структура. Конфігуруючи даний файл можна змінювати якість та поведінку нейронної мережі.

Нейронна мережа може розпізнати наступні класи:

- information-priority-road_end;
- information-priority-road;
- prohibited-access-entry;
- prohibited-access;
- prohibited-action-overtaking;
- warning-crossroad-give-way;
- warning-crossroad-stop;
- information-parking;
- prohibited-parking-stopping;
- information-crossing-pedestrian;
- prohibited-road-narrowing;
- information-motorway_end;
- information-motorway;
- warning-pedestrian-crossing;
- prohibited-action-speed;

- prohibited-parking;
- warning;
- information-road-narrowing;
- warning-traffic-lights.

Конфігураційний файл було основано на базі конфігурації TinyYOLOv3. Він є достатньо насичений шарами і в той же ж час легким, тому нейронні мережі основані на ньому є швидкими і продуктивними. А вихідні ваги для мережі мають невеликий об’єм.

Нижче наведені основні параметри за допомогою яких налаштовується нейронна мережа:

- batch=1
- subdivisions=1
- width=832
- height=832
- channels=3
- momentum=0.9
- decay=0.0005
- angle=0
- saturation = 1.5
- exposure = 1.5
- hue=.1
- flip=0
- learning_rate=0.001
- burn_in=1000
- max_batches = 38000
- policy=steps
- steps=304000,34200
- scales=.1,.1

Також на ефективність нейронної мережі впливає розташування і кількість шарів[1]. Для реалізації нейронної мережі для розпізнавання дорожніх знаків було обрано структуру згорткової нейронної мережі. Її структура має наступний вигляд (Рисунок 4.3):

[convolutional]

[maxpool]

[convolutional]

[maxpool]

[convolutional]

[maxpool]

[convolutional]

[maxpool]

[convolutional]

[maxpool]

[convolutional]

[convolutional]

[convolutional]

[convolutional]

[yolo]

[convolutional]

[upsample]

[convolutional]

[convolutional]

[yolo]

Рисунок 4.3 — Структура нейронної мережі

Такий набір шарів є ефективним для розв’язання такої задачі як розпізнавання дорожніх знаків. Нейронна мережа одночасно і ефективна, і не перенасичена лишніми шарами.

Convolutional layer (шар згортки) — це основний блок згорточної нейронної мережі. Шар згортки включає в себе для кожного каналу свій фільтр, ядро згортки якого обробляє попередній шар по фрагментам (підсумовуючи результати матричного множення для кожного фрагмента)(Рисунок 4.4). Вагові коефіцієнти ядра згортки (невеликої матриці) невідомі і встановлюються в процесі навчання.

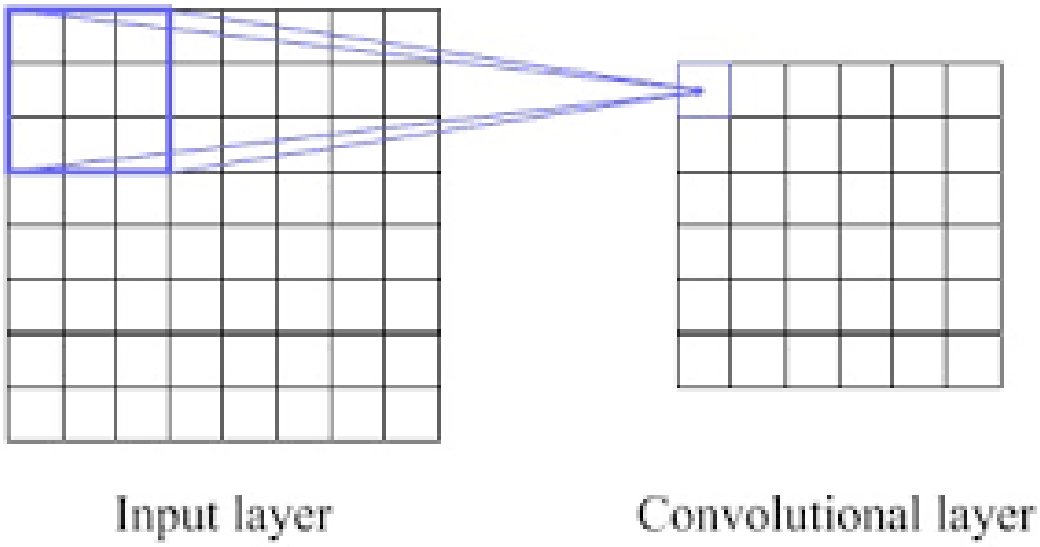

Рисунок 4.4 — Зображення шару згортки

Особливістю шару згортки є порівняно невелика кількість параметрів, які встановлюються при навчанні.

Так наприклад, якщо вихідне зображення має розмірність 100 × 100 пікселів по трьом каналам (це значить 30000 вхідних нейронів), а шар згортки використовує фільтри з ядром 3x3 пікселя з виходом на 6 каналів, тоді в процесі навчання визначається тільки 9 ваг ядра, однак по всім сполученням каналів, тобто 9 × 3 × 6 = 162, в такому випадку даний шар вимагає знаходження тільки 162 параметрів, що істотно менше кількості шуканих параметрів повно нейронної мережі.

Maxpool layer — шар пулінга представляє собою нелінійне ущільнення карти ознак, при цьому група пікселів (звичайно розмір 2 × 2) стискується до одного пікселя, проходячи нелінійне перетворення (Рисунок 4.5). Найбільш популярна при цьому функція максимуму.

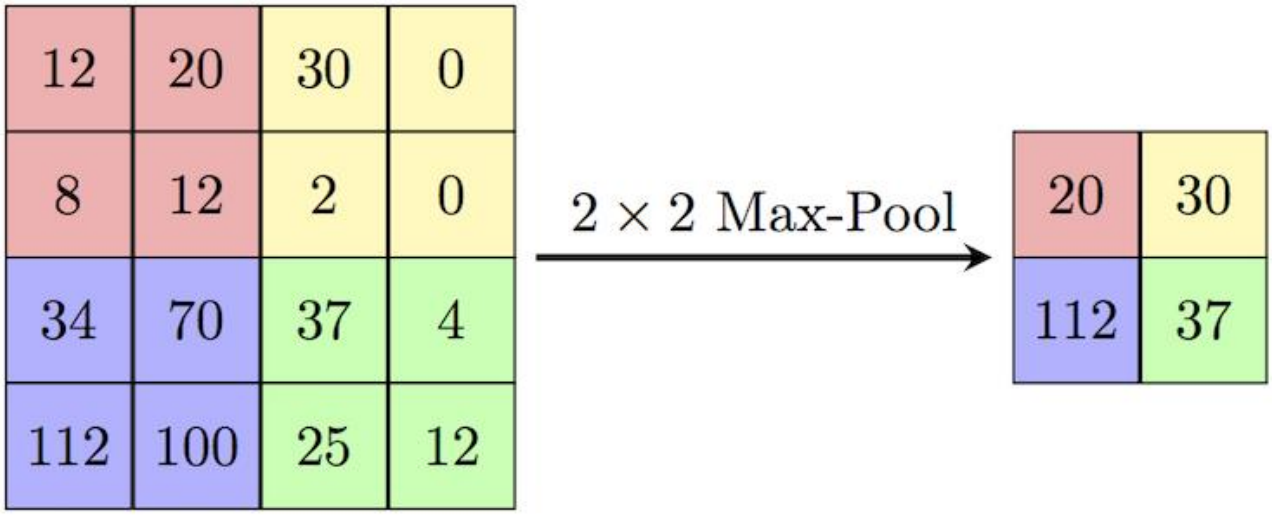

Рисунок 4.5 — Зображення пулінгу з функцією максимума

Перетворення зачіпає прямокутники або квадрати, які не пересікаються кожний з яких стискується до одного пікселя, при цьому вибирається піксель, що має максимальне значення. Операція пулінга дозволяє істотно зменшити простір об'єму зображення.

Пулінг інтерпретується так: якщо на попередній операції вже були виявлені деякі визнані ознаки, то для подальшої обробки даних такого детального зображення не потрібно. До того ж фільтрація вже непотрібних деталей допомагає не перенавчатись нейронній мережі.

Як правило, шар пулінга вставляється між шарами згортки. Крім пулінга з функцією максимума можна використовувати і інші функції - наприклад, середнє значення або L2-нормування.

Upsample layer — цей шар використовується для підвищення роздільної здатності нейронної мережі і є протилежним до пулінгу (Рисунок 4.6).

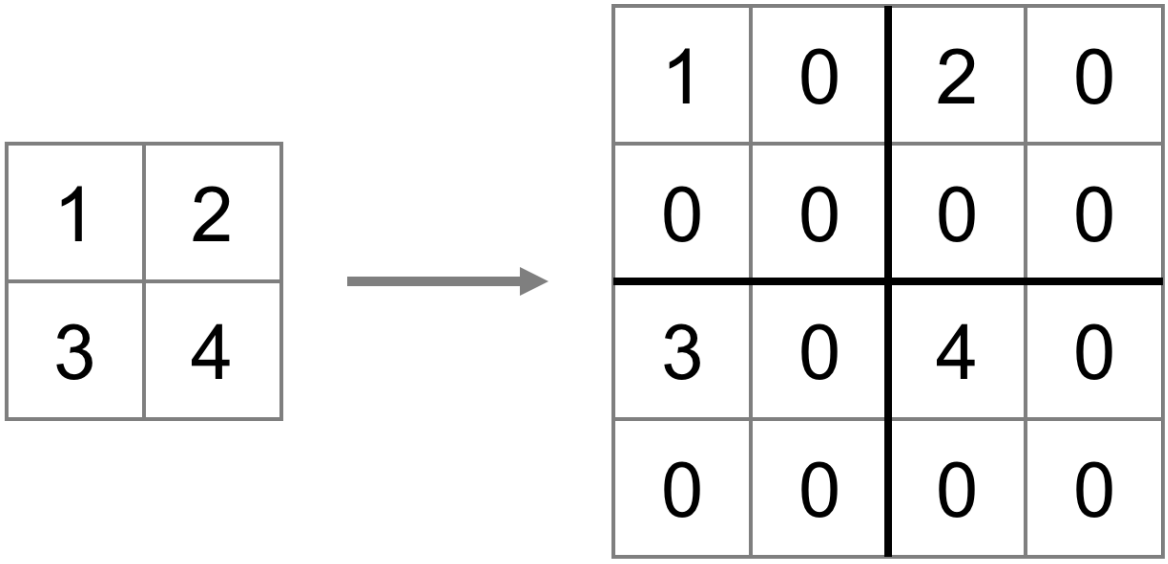

Рисунок 4.6 — Зображення upsample

Як видно на рисунку 4.6 розмір збільшився вдвічі і на нові місця були записані нулі.

4.3. Опис створення датасету для нейронної мережі

Процес створення датасету є доволі відповідальним етапом в створенні нейронної мережі. Для початку було завантажено декілька відео з відеореєстраторів. Потім за допомогою програми VLC з них було отримано багато різноманітних зображень з фоном дороги чи міста.

Дорожні знаки були завантажені з інтернету і потребували попередньої обробки. А саме викривлення, зменшення або збільшення контрастності, яркості, різкості і багато інших параметрів. Для цього я використовував Python та бібліотеку Imgaug[6].

Imgaug[6] - це бібліотека для збільшення зображень в експериментах з машинного навчання. Вона підтримує широкий спектр методів збільшення, дозволяє

легко комбінувати їх і виконувати їх у випадковому порядку або на декількох ядрах процесора, має простий, але потужний інтерфейс і може не тільки збільшувати зображення, але й ключові точки / орієнтири, обмежувальні поля, теплові карти і карти сегментації.

Розглянемо деформації на прикладі дорожнього знаку стоп, який зображений на Рисунку 4.5

Рисунок 4.5 — Знак стоп без змін

На Рисунках 4.6, 4.7 та 4.8 наведено дорожній знак після різних деформацій, таких як:

- Афінне перетворення;
- Часткове пошкодження;
- Кусково-афінне перетворення.

Рисунок 4.6 — Афінне перетворення

Рисунок 4.7 — Часткове пошкодження

Рисунок 4.8 — Кусково-афінне перетворення

Таким чином було підвищено точність нейронної мережі, завдяки різноманітності форми дорожнього знаку.

Після генерації дорожніх знаків потрібно об’єднати фон дороги (Рисунок 4.9) і згенеровані знаки в одне зображення. Для всіх цих дій були завантажені та модифіковані або написані відповідні скрипти на мові програмування Python[4].

Рисунок 4.9 — Фон дороги

Головне, щоб на фоні не було не відмічених дорожніх знаків, тому що нейронна мережа буде навчатись неправильно.

На Рисунку 4.10 зображено вихідне зображення, яке є результатом об’єднання фону дороги та згенерованого дорожнього знаку.

Рисунок 4.10 — Згенероване зображення

Таким способом генерується весь датасет для нейронної мережі. Потім він ділиться на тренувальний і валідаційний.

4.4. Процес навчання нейронної мережі

Для початку потрібно правильно сконфігурувати файл конфігурації для навчання. А саме змінити такі поля:

- batch=64;
- subdivisions=8;
- max_batches= classes*2000, де classes=кількість класів для розпізнавання. Тобто в нас max_batches = 38000;
- steps на 80% і 90% від max_batches, в нашому випадку steps=304000,34200;
- classes=19 в кожному [yolo] шарі;

- filters=(classes + 5) x 3 у 3 [convolutional] шарах перед кожним [yolo] шаром, в нашому випадку filters=72;

(Взагалі фільтри залежать від класів, координатів і кількості масок, тобто фільтри = (класи + корди + 1) * <кількість масок>, де маска - індекси якорів. + 1) * num).

Створено файл road-signs.names з іменами об'єктів — кожен у новому рядку.

Створіть файл road-signs.data, що містить:

- класи = 19
- train = data / road_signs_train.txt
- valid = data / road_signs_test.txt
- names = data / road_signs.names
- backup = backup /

В файлах road_signs_train.txt та _signs_test.txt записані шляхи до відповідних текстових файлів, які характеризуються зображення з датасету. Цей .txt файл для кожного .jpg файлу містить в собі: номер об'єкта і координати об'єкта на цьому зображенні, для кожного об'єкта у новому рядку:

<об'єкт-клас> <x_center> <y_center> <ширина> <висота>,

Де:

<object-class> — номер об’єкта-класа;

<x_center> <y_center> <width> <height> — значення float, відносна ширина та висота зображення, яке може бути рівним (від 0,0 до 1,0)

наприклад: <x> = <absolute_x> / <image_width> або <height> = <absolute_height> / <image_height>

на увазі: <x_center> <y_center> — центр прямокутника (не верхній лівий кут)

Наприклад для 1.jpg було створено 1.txt, що містить:

1 0,5 0,5 1,0 1,0

Було створено файл road_signs_train.txt у з іменами файлів зображень, кожне ім'я файлу в новому рядку, з шляхом відносно darknet:

data / obj / 1.jpg

data / obj / 2.jpg

data / obj / 3.jpg

Попередньо було завантажено підготовлені ваги для згорткових шарів (154 МБ).

Для навчання в Linux використовуємо команду: ./darknet detector train data / road-signs.data road-signs-obj-tiny.cfg darknet53.conv.74

Для навчання з mAP (середня середня точність) розрахунок для кожних 4 Епох (встановіть valid = road_signs_valid.txt або road_signs_train.txt в файлі road-signs.data) і запустіть: darknet detector train data/ obj.data yolo-obj.cfg darknet53 .conv.74 –map.

На рисунку 4.9 можна бачити процес тренування.

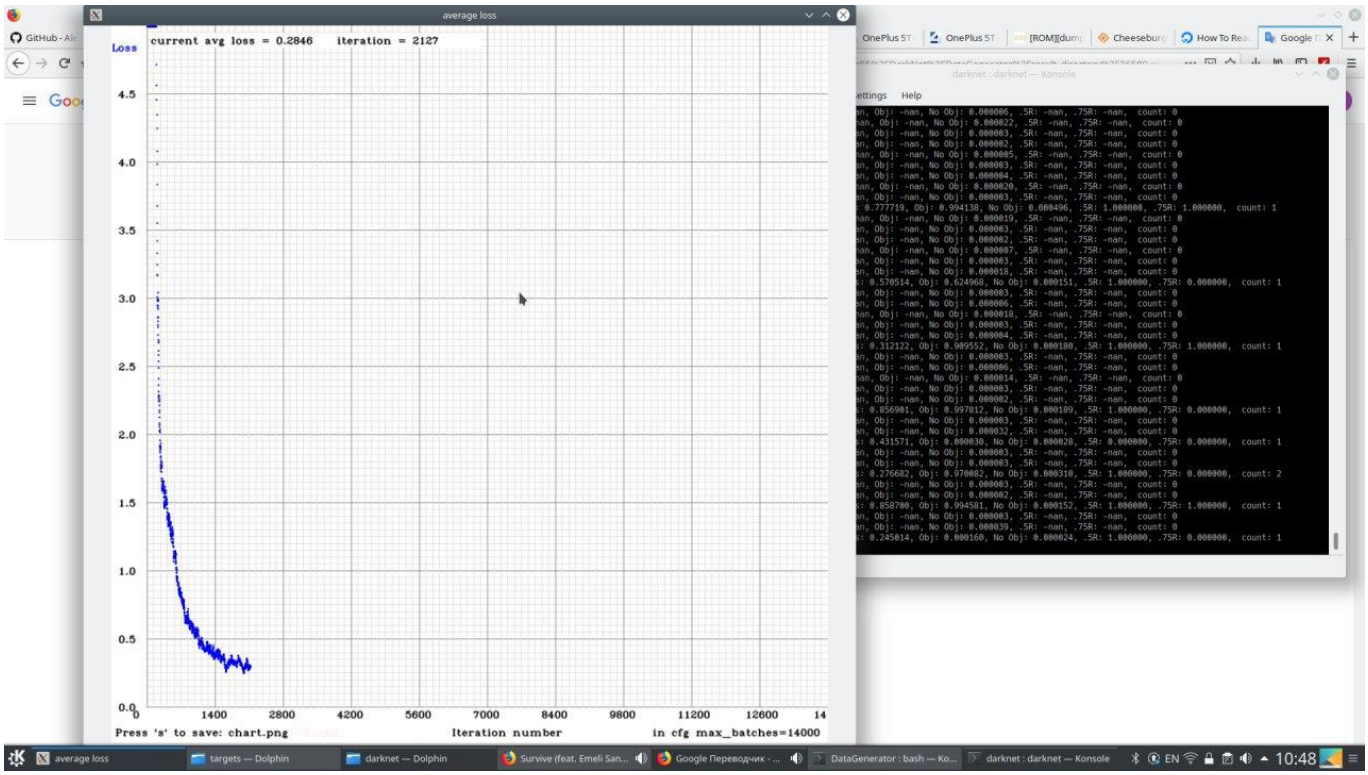

Рисунок 4.9 — Процес тренування

Синім відображається похибка розпізнавання нейронної мережі. Чим менше значення, тим краще вона працює.

Після завершення навчання отримайте результат yolo-obj_final.

Після кожних 100 ітерацій ви можете зупинитися і пізніше почати навчання з цього моменту. Наприклад, після 2000 ітерацій ви можете зупинити тренування, а пізніше просто почати тренування з того самого місця.

Якщо під час тренування ви бачите Nan значення для середньої похибки — тоді навчання йде неправильно, але якщо Nan є в деяких інших лініях — тоді навчання йде добре.

4.6. Висновки до розділу

У розділі було описано програмну реалізацію розроблюваної системи, а також її функціональність. Приведено структуру нейронної мережі та описано процес створення датасету для навчання та валідації.

5. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

5.1. Інсталяція та системні вимоги

Після інсталяції програмного забезпечення потрібно записати в налаштування клієнта IP адресу серверу. Після цього налаштування вважається виконаним і можна переходити до використання продукту.

5.2. Інструкція з використання програмного продукту

Перш за все потрібно запустити сервер. Тут можна спостерігати за підключенням клієнта та його активністю. Він виглядає наступним чином:

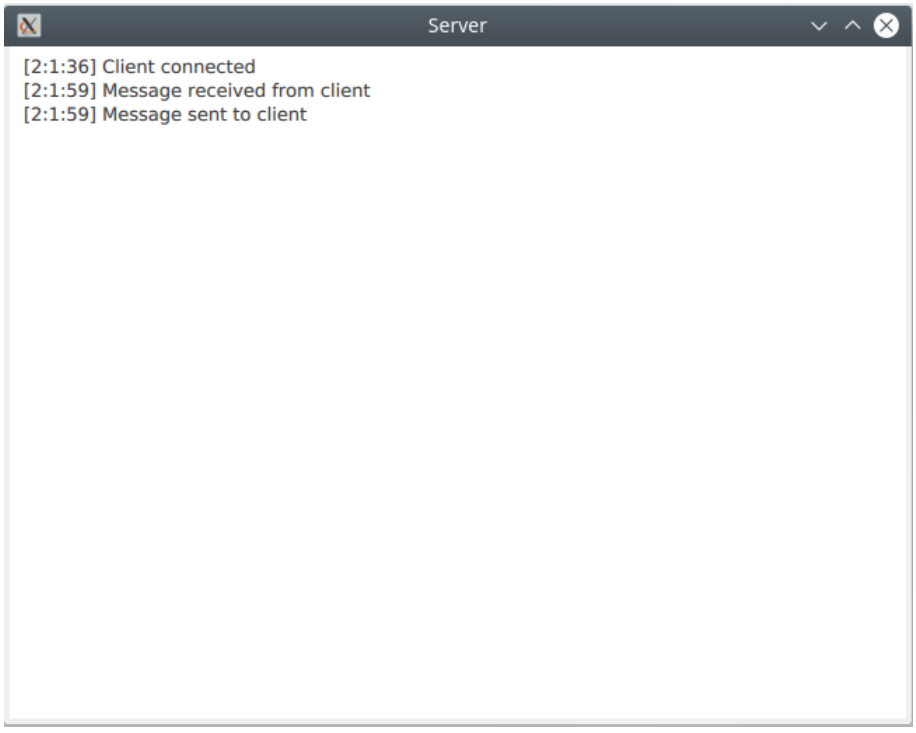

Рисунок 5.1 — Вікно серверу

При першому запуску мобільного додатку потрібно надати права на використання камери пристрою та перевірити чи активний сервер, щоб почати працювати в системі. Інтерфейс програми інтуїтивний та простий. На екрані знаходиться одна кнопка яка відповідає за фотографування. На рисунку 5.2 зображений мобільний додаток.

Рисунок 5.2 — Екран мобільного додатку

Після того, як користувач сфотографує дорожній знак, він отримує у відповідь

зображення з розпізнаним дорожнім знаком (Рисунок 5.3).

Рисунок 5.3 — Екран мобільного додатку після розпізнавання

При успішному розпізнаванні на зображенні з’явиться контур та підпис навколо знаку. Також внизу екрана з’являться мініатюри розпізнаних знаків при натисканні на які можна детально прочитати опис знаку. Для повернення до камери потрібно просто натиснути по зображенню.

У випадку невдачі користувач матиме змогу зробити фото ще раз.

Рисунок 5.4 — Детальний опис розпізнаного дорожнього знаку

Детальний опис відображається у вигляді спливаючого вікна. Він містить фото знаку та його детальний опис. Для його закриття потрібно натиснути поза його межами.

ВИСНОВКИ

У ході аналізу існуючого програмного забезпечення розпізнавання дорожніх знаків на основі нейронної мережі було досліджено системи, які відповідають для вирішення поставленої задачі. Аналіз дав чітко зрозуміти, що існуючі системи не підходять для повного вирішення поставленої задачі.

Реалізовано систему, яка дозволяє розпізнавати дорожні знаки та інформувати користувача, що це за знак.

Проведено огляд різних методів та засобів розробки програмного продукту. Обґрунтовано вибір створення програмного продукту, який реалізовано за допомогою фреймворку Qt на мові C++. Це дає змогу полегшити процес інтегрування даної системи до будь-якого іншого автомобільного рішення.

За результатами виконаних тестових знімків підтверджена коректність розпізнаних зображень, отже система відповідає поставленим вимогам.

Користувачами даної системи можуть бути як звичайні користувачі, які хочуть дізнатись назву того чи іншого знаку, так і різноманітні автомобільні розробники, які хочуть інтегрувати дану систему до своїх передових рішень. Серверна частина програмного забезпечення може бути завантажена на Linux або Windows, а клієнтська частина на Android, Linux або Windows.

Отже, під час практики було покращено знання в різних напрямках, що використовуються під час розробки програмного забезпечення. Також було вивчено багато нових технологій та можливостей, які в майбутньому будуть корисні, як розробнику програмного забезпечення. Також було створено декілька тестових нейронних мереж для розпізнавання інших об’єктів, які лягли в основу розробленого програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Windows and Linux version of Darknet Yolo v3 & v2 Neural Networks for object detection (Tensor Cores are used) http://pjreddie.com/darknet/ [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/alexeyab/darknet.

2. YOLO: Real-Time Object Detection [Електронний ресурс] – Режим доступу до ресурсу: https://pjreddie.com/darknet/yolo/.

3. Card detection using YOLO on Android [Електронний ресурс] – Режим доступу до ресурсу: https://www.tooploox.com/blog/card-detection-using-yolo-on-android.

4. Datagenerator [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/fastant/datagenerator.

5. Николенко С., Кадурин А., Архангельская Е.. Глубокое обучение. - Спб.: Питер, 2018. - 180 с.: ил. – ("Серия Библиотека программиста”).

6. Alexander J. Imgaug [Електронний ресурс] / Jung Alexander – Режим доступу до ресурсу: https://imgaug.readthedocs.io/en/latest/index.html.

7. Recognising Traffic Signs With 98% Accuracy Using Deep Learning [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/recognizing-traffic-signs-with-over-98-accuracy-using-deep-learning-86737aedc2ab.

ДОДАТОК А

Система розпізнавання дорожніх знаків на основі нейронної мережі

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ51193_ 18Б 12-2

Аркушів 1

Київ — 2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТІ51193_18Б 79-1	Записка Фастовець.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ51193_18Б 12-1	server.pro	Серверний додаток
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ51193_ 18Б 12-2	client.pro	Клієнтський додаток

ДОДАТОК Б

Система розпізнавання дорожніх знаків на основі нейронної мережі

Текст програмного модуля

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ51193_ 18Б 12-2

Аркушів 8

Київ — 2019

Server:

//detector.h

```
#ifndef SIGN_DETECTOR_H
#define SIGN_DETECTOR_H

#include <QObject>
#include <QImage>
#include <QQmlEngine>

#include "include/yolo_v2_class.hpp"

class SignDetector : public QObject
{
    Q_OBJECT
public:
    explicit SignDetector(QObject *parent = nullptr);

    QString config() const;
    void setConfig(const QString &config);

    QString weights() const;
    void setWeights(const QString &weights);

    QString namesfile() const;
    void setNamesfile(const QString &namesfile);

    void draw_boxes(cv::Mat mat_img, std::vector<bbox_t> result_vec, std::vector<std::string> obj_names,
                    int current_det_fps = -1, int current_cap_fps = -1);

    void qimage_to_mat(const QImage& image, cv::OutputArray out);

    void mat_to_qimage(cv::InputArray image, QImage& out);

    std::vector<std::string> objects_names_from_file(std::string const filename) {
        std::ifstream file(filename);
        std::vector<std::string> file_lines;
        if (!file.is_open()) return file_lines;
        for(std::string line; getline(file, line);) file_lines.push_back(line);
        std::cout << "object names loaded \n";
        return file_lines;
    }
signals:
    void imageRecognized(QString imageData);
    void sendDetectedSignsStr(QString detectedSignsStr);

public slots:
    void test_detector(QString image_str);

private:
    QString m_config;
    QString m_weights;
    QString m_namesfile;

    std::shared_ptr<Detector> detector;
    std::vector<std::string> obj_names;
};

#endif // SIGN_DETECTOR_H
```

//detector.cpp

```
#include "detector.h"
#include "include/darknet.h"
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
#include <queue>
#include <fstream>
```

```
#include <thread>
#include <future>
#include <atomic>
#include <mutex>        // std::mutex, std::unique_lock
#include <cmath>
#include <QBuffer>
#include <QDebug>
#include <QByteArray>
#include <QTemporaryFile>
#include <QQuickImageProvider>

SignDetector::SignDetector(QObject *parent) : QObject(parent)
{
    setConfig(":/data_for_detector/road-signs-obj-tiny.cfg");
    setWeights(":/data_for_detector/road-signs-obj-tiny.weights");
    setNamesfile(":/data_for_detector/road_signs.names");

    std::string  names_file = "/tmp/names_file";
    std::string  cfg_file = "/tmp/cfg_file";
    std::string  weights_file = "/tmp/weights_file";

    QFile::copy(m_namesfile, names_file.c_str());
    QFile::copy(m_config, cfg_file.c_str());
    QFile::copy(m_weights, weights_file.c_str());

    detector.reset(new Detector(cfg_file, weights_file));

    obj_names = objects_names_from_file(names_file);

}

QString SignDetector::config() const
{
    return m_config;
}

void SignDetector::setConfig(const QString &config)
{
    m_config = config;
}

QString SignDetector::weights() const
{
    return m_weights;
}

void SignDetector::setWeights(const QString &weights)
{
    m_weights = weights;
}

void SignDetector::draw_boxes(cv::Mat mat_img, std::vector<bbox_t> result_vec, std::vector<std::string> obj_names,
                int current_det_fps, int current_cap_fps)
{
    QStringList list;

    for (auto &i : result_vec) {
        cv::Scalar color = obj_id_to_color(i.obj_id);
        cv::rectangle(mat_img, cv::Rect(i.x, i.y, i.w, i.h), color, 2);

        if (obj_names.size() > i.obj_id) {
            std::string obj_name = obj_names[i.obj_id];
            list.append(QString::fromStdString(obj_name));

            if (i.track_id > 0) obj_name += " - " + std::to_string(i.track_id);
            cv::Size const text_size = getTextSize(obj_name, cv::FONT_HERSHEY_COMPLEX_SMALL, 1.2, 2, 0);
            int max_width = (text_size.width > i.w + 2) ? text_size.width : (i.w + 2);
            max_width = std::max(max_width, (int)i.w + 2);
            //max_width = std::max(max_width, 283);
            std::string coords_3d;

            if (!std::isnan(i.z_3d)) {
                std::stringstream ss;
                ss << std::fixed << std::setprecision(2) << "x:" << i.x_3d << "m y:" << i.y_3d << "m z:" << i.z_3d << "m ";
                coords_3d = ss.str();
                cv::Size const text_size_3d = getTextSize(ss.str(), cv::FONT_HERSHEY_COMPLEX_SMALL, 0.8, 1, 0);
                int const max_width_3d = (text_size_3d.width > i.w + 2) ? text_size_3d.width : (i.w + 2);
                if (max_width_3d > max_width) max_width = max_width_3d;
```

```
            }

            cv::rectangle(mat_img, cv::Point2f(std::max((int)i.x - 1, 0), std::max((int)i.y - 35, 0)),
                          cv::Point2f(std::min((int)i.x + max_width, mat_img.cols - 1), std::min((int)i.y, mat_img.rows - 1)),
                          color, CV_FILLED, 8, 0);
            putText(mat_img, obj_name, cv::Point2f(i.x, i.y - 16), cv::FONT_HERSHEY_COMPLEX_SMALL, 1.2, cv::Scalar(0, 0, 0), 2);
            if(!coords_3d.empty()) putText(mat_img, coords_3d, cv::Point2f(i.x, i.y-1), cv::FONT_HERSHEY_COMPLEX_SMALL, 0.8, cv::Scalar(0, 0, 0), 1);
        }
    }

    QString detected_signs = "signs:" + list.join(",");

    emit sendDetectedSignsStr( detected_signs);

    if (current_det_fps >= 0 && current_cap_fps >= 0) {
        std::string fps_str = "FPS detection: " + std::to_string(current_det_fps) + "   FPS capture: " + std::to_string(current_cap_fps);
        putText(mat_img, fps_str, cv::Point2f(10, 20), cv::FONT_HERSHEY_COMPLEX_SMALL, 1.2, cv::Scalar(50, 255, 0), 2);
    }
}

void SignDetector::qimage_to_mat(const QImage &image, cv::OutputArray out) {

    switch(image.format()) {
    case QImage::Format_Invalid:
    {
        cv::Mat empty;
        empty.copyTo(out);
        break;
    }
    case QImage::Format_RGB32:
    {
        cv::Mat view(image.height(),image.width(),CV_8UC4,(void *)image.constBits(),image.bytesPerLine());
        view.copyTo(out);
        break;
    }
    case QImage::Format_RGB888:
    {
        cv::Mat view(image.height(),image.width(),CV_8UC3,(void *)image.constBits(),image.bytesPerLine());
        cvtColor(view, out, cv::COLOR_RGB2BGR);
        break;
    }
    default:
    {
        QImage conv = image.convertToFormat(QImage::Format_ARGB32);
        cv::Mat view(conv.height(),conv.width(),CV_8UC4,(void *)conv.constBits(),conv.bytesPerLine());
        view.copyTo(out);
        break;
    }
    }
}

void SignDetector::mat_to_qimage(cv::InputArray image, QImage &out)
{
    switch(image.type())
    {
    case CV_8UC4:
    {
        cv::Mat view(image.getMat());
        QImage view2(view.data, view.cols, view.rows, view.step[0], QImage::Format_ARGB32);
        out = view2.copy();
        break;
    }
    case CV_8UC3:
    {
        cv::Mat mat;
        cvtColor(image, mat, cv::COLOR_BGR2BGRA); //COLOR_BGR2RGB doesn't behave so use RGBA
        QImage view(mat.data, mat.cols, mat.rows, mat.step[0], QImage::Format_ARGB32);
        out = view.copy();
        break;
    }
    case CV_8UC1:
    {
        cv::Mat mat;
        cvtColor(image, mat, cv::COLOR_GRAY2BGRA);
        QImage view(mat.data, mat.cols, mat.rows, mat.step[0], QImage::Format_ARGB32);
        out = view.copy();
        break;
    }
```

```
    default:
    {
        break;
    }
    }
}

void SignDetector::test_detector(QString image_str)
{
    QByteArray array;
    array.append(image_str);

    QImage image;
    image.loadFromData(QByteArray::fromBase64(array));

    try {
        cv::Mat mat_img;
        qimage_to_mat(image, mat_img);

        //      auto start = std::chrono::steady_clock::now();
        std::vector<bbox_t> result_vec = detector->detect(mat_img);
        //      auto end = std::chrono::steady_clock::now();
        //      std::chrono::duration<double> spent = end - start;
        //      std::cout << " Time: " << spent.count() << " sec \n";

        draw_boxes(mat_img, result_vec, obj_names);

        mat_to_qimage(mat_img, image);

        array.clear();
        QBuffer bu(&array);

        bu.open(QIODevice::WriteOnly);
        image.save(&bu, "JPG");

        QString imgBase64 = QString::fromLatin1(array.toBase64().data());
        emit imageRecognized(imgBase64);
    }
    catch (std::exception &e) { std::cerr << "exception: " << e.what() << "\n"; getchar(); }
    catch (...) { std::cerr << "unknown exception \n"; getchar(); }
}

QString SignDetector::namesfile() const
{
    return m_namesfile;
}

void SignDetector::setNamesfile(const QString &namesfile)
{
    m_namesfile = namesfile;
}
```

//main.qml

```
import QtQuick 2.11
import QtQuick.Controls 2.5
import QtMultimedia 5.12
import QtWebSockets 1.1

ApplicationWindow {
    visible: true
    width: 640
    height: 480
    title: qsTr("Server")

    function appendMessage(message) {
        var currentDate = new Date();
        var datetime = "[" + currentDate.getHours() + ":"
                + currentDate.getMinutes() + ":" + currentDate.getSeconds() + "]";
        textArea.append(datetime + " " + message);
    }

    WebSocketServer {
        id: server

        property var client: null
```

```
        listen: true
        host: "10.42.0.1"
        port: 1234
        onClientConnected: {
            appendMessage("Client connected");
            client = webSocket;
            webSocket.onTextMessageReceived.connect(function(message) {
                appendMessage("Message received from client");
                detector.test_detector(message);
            });
        }
        onErrorStringChanged: {
            appendMessage(qsTr("Server error: %1").arg(errorString));
        }
    }

    Connections {
        target: detector

        onImageRecognized: {
            server.client.sendTextMessage(imageData)
            appendMessage("Message sent to client");
        }

        onSendDetectedSignsStr: {
            server.client.sendTextMessage(detectedSignsStr)
            appendMessage("Message sent to client");
        }
    }

    TextArea {
        id: textArea

        anchors.fill: parent
    }
}
```

Client:

//imageconvertor.h

```
#ifndef IMAGECONVERTER_H
#define IMAGECONVERTER_H

#include <QObject>
#include <QQmlEngine>
#include <QSize>

class ImageConverter : public QObject
{
    Q_OBJECT
public:
    explicit ImageConverter(QQmlEngine *engine, QObject *parent = nullptr);

signals:

public slots:
    QString toBase64(QString image_str, QSize resolution);
private:
    QQmlEngine *engine;
};

#endif // IMAGECONVERTER_H
```

//imageconvertor.cpp

```
#include "imageconverter.h"
#include <QDebug>
#include <QQuickImageProvider>
#include <QImage>
#include <QBuffer>

ImageConverter::ImageConverter(QQmlEngine *engine, QObject *parent) : QObject(parent), engine(engine)
{
```

```
}

QString ImageConverter::toBase64(QString image_str, QSize resolution)
{
    QQuickImageProvider *image_provider = static_cast<QQuickImageProvider *>(engine->imageProvider("camera"));

    QImage image(image_provider->requestImage(image_str.replace("image://camera/", ""), &resolution, resolution));

    QByteArray ba;
    QBuffer bu(&ba);

    bu.open(QIODevice::WriteOnly);
    image.save(&bu, "PNG");

    QString imgBase64 = QString::fromLatin1(ba.toBase64().data());
    return imgBase64;
}
```

//main.qml

```
import QtQuick 2.12
import QtQuick.Controls 2.5
import QtQuick.Layouts 1.12
import QtWebSockets 1.0
import QtMultimedia 5.12

ApplicationWindow {
    id: root

    property var detectedSigns: []

    visible: true
    width: 360
    height: 640

    function appendMessage(message) {
        console.log(message);
    }

    WebSocket {
        id: socket

        url: "ws://10.42.0.1:1234"
        active: true
        onTextMessageReceived: {
            container.visible = true;

            if (message.search("signs:") !== -1) {
                var detectedSignsStr = message.replace("signs:", "");
                if (detectedSignsStr.length !== 0) {
                    detectedSigns = detectedSignsStr.split(",");
                }
            }
            else {
                image.imageData = message;
            }
        }

        onStatusChanged: {
            if (socket.status == WebSocket.Error) {
                appendMessage(qsTr("Client error: %1").arg(socket.errorString));
            } else if (socket.status == WebSocket.Closed) {
                appendMessage(qsTr("Client socket closed."));
            }
        }
    }

    Camera {
        id: camera

        viewfinder.resolution: Qt.size(1280, 720)

        imageCapture {
            resolution: Qt.size(1280, 720)
            onImageCaptured: {
                socket.active = true;
                socket.sendTextMessage(imageConvertor.toBase64(preview, camera.imageCapture.resolution));
```

```
        }
    }
}

VideoOutput {
    id: viewfinder

    anchors.fill: parent

    source: camera
    autoOrientation: true

    MouseArea {
        anchors.fill: parent
        onClicked: {
            if (camera.lockStatus == Camera.Unlocked)
                camera.searchAndLock();
            else
                camera.unlock();
        }
    }

    RoundButton {
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.bottom: parent.bottom
        anchors.bottomMargin: width / 4

        width: parent.width / 5
        height: width

        onClicked: {
            camera.imageCapture.capture();
        }
    }
}

Item {
    id: container

    anchors.fill: parent

    visible: false

    Image {
        id: image

        anchors {
            left: parent.left
            right: parent.right
            top: parent.top
            bottom: parent.bottom
        }

        property string imageData: ""

        source: "" + imageData

        visible: imageData.length > 0

        MouseArea {
            anchors.fill: parent
            onClicked: {
                image.imageData = "";
                container.visible = false;
            }
        }
    }


    ListView {
        id: listView

        anchors.left: parent.left
        anchors.right: parent.right
        anchors.bottom: parent.bottom

        orientation: ListView.Horizontal
```

```
        height: detectedSigns.length > 0 ? (parent.width / 5) : 0

        model: detectedSigns

        delegate: Image {
            height: parent.height
            width: height

            source: "/road_signs/" + detectedSigns[index] + ".png"

            MouseArea {
                anchors.fill: parent

                onClicked: {
                    listView.currentIndex = index;
                    contentDialog.open();
                }
            }
        }
    }

    Dialog {
        id: contentDialog

        x: (parent.width - width) / 2
        y: (parent.height - height) / 2
        width: Math.min(parent.width, parent.height) / 4 * 3
        height: Math.max(parent.width, parent.height) / 4 * 3
        parent: Overlay.overlay

        modal: true

        Flickable {
            id: flickable
            clip: true
            anchors.fill: parent
            contentHeight: column.height

            Column {
                id: column
                spacing: 20
                width: parent.width

                Image {
                    id: logo
                    width: parent.width / 2
                    anchors.horizontalCenter: parent.horizontalCenter
                    fillMode: Image.PreserveAspectFit
                    source: "/road_signs/" + detectedSigns[listView.currentIndex] + ".png"
                }

                Label {
                    width: parent.width

                    wrapMode: Label.Wrap
                }
            }

            ScrollIndicator.vertical: ScrollIndicator {
                parent: contentDialog.contentItem
                anchors.top: flickable.top
                anchors.bottom: flickable.bottom
                anchors.right: parent.right
                anchors.rightMargin: -contentDialog.rightPadding + 1
            }
        }
    }
  }
}
```

ДОДАТОК В

Система розпізнавання дорожніх знаків на основі нейронної мережі

Опис програмного модуля

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ51193_ 18Б 12-2

Аркушів 7

Київ — 2019

АНОТАЦІЯ

У додатку надана інформація про програмну частину системи розпізнавання дорожніх знаків на основі нейронної мережі.

Програмний продукт являє собою програму(сервер) для ПК на базі Linux та клієнтський додаток для Android. Для створення було використано фреймворк Qt.

Серверний додаток отримує вхідне зображення від клієнта та передеє його для розпізнавання нейронній мережі. Після розпізнавання сервер відправляє результат до клієнта.

Клієнтський додаток надає можливість сфотографувати дорожній знак, отримати розпізнане зображення та дізнатись інформацію про розпізнаний знак.

ЗМІСТ

Анотація 63
Зміст 64
Функціональне призначення 65
Опис логічної структури 66
Використовувані технічні засоби 67
Виклик і завантаження 68
Вхідні та вихідні дані 69

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Основним призначенням даного програмного забезпечення є надання користувачу можливості швидкого розпізнання дорожніх знаків і збереження інформації про них.

Розроблений програмні модулі продукту мають такі функції:

- розпізнати дорожній знак;
- зробити фото;
- показати детальний опис знаку;
- зберегти зображення;

ОПИС СТРУКТУРИ СИСТЕМИ

Програмне забезпечення складається із двох основних частин, що взаємлдіють між собою: серверний і клієнтьский додаток, між якими відбувається взаємодія за допомогою WebSocket технології.

Задачу сервера можна розділити на дві частини:

- отримання зображення від клієнту та їх відправка;
- розпізнавання дорожніх знаків за допомогою нейронної мережі.

Программа має загалом такі основні функції:

- mat_to_qimage();
- qimage_to_mat();
- draw_boxes();
- test_detector().

Функції mat_to_qimage() та qimage_to_mat () забезпечують коректне ковертування зображеня між двома форматами: cv::Mat та QImage.

Функція draw_boxes() малює контури навколо розпізнаного дорожнього знаку та пише його назву.

Функція test_detector() приймає на вхід зображення у base64 форматі конвертує в правильний формат, розпізнає дорожній знак і повертає результат.

Клієнтський додаток складається із двох частин:

• Інтерфейс користувача;

• WebSocket клієнт.

Для передачі зображення на сервер, воно конвертується в base64 формат.

Для цього було написано клас ImageConvertor, який містить в собі функцію toBase64, яка на вхід приймає зображення та роздільну здатність фото, а повертає зображення у base64 форматі.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для створення системи було використано:

- Qt;
- Yolo;
- CUDA;
- OpenCV;
- Python;
- QtCreator.

Для запуску серверного додатку потрібний ПК з відеокартою Nvidia, яка підтримує Cuda обчислення, а також встановлена бібліотека OpenCV.

Для запуску клієнтського додатку потрібний смартфон на Android 7.0 і вище.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Серверний додаток не потрібно встановлювати. Достатньо запустити виконуваний файл програмного додатку. Клієнтський додаток потрібно завантажити та інсталювати.

ВХІДНІ ТА ВИХІДНІ ДАНІ

Дана система на вхід приймає зображення, фото оброблюється і відправляється до нейронної мережі, вона розпізнає його і повертає користувачу. Отримане зображення зберігається на пристрої користувача.